

A graph neural network and multi-task learning-based decoding algorithm for enhancing *XZZX* code stability in biased noise

Bo Xiao(肖博), Zai-Xu Fan(范在旭), Hui-Qian Sun(孙汇倩), Hong-Yang Ma(马鸿洋), and Xing-Kui Fan(范兴奎)

Citation: Chin. Phys. B, 2025, 34 (**5**): 050306. DOI: 10.1088/1674-1056/adbadb
Journal homepage: http://cpb.iphy.ac.cn; http://iopscience.iop.org/cpb

What follows is a list of articles you may be interested in

Global receptive field transformer decoder method on quantum surface code data and syndrome error correction

Ao-Qing Li(李熬庆), Ce-Wen Tian(田策文), Xiao-Xuan Xu(徐晓璇), Hong-Yang Ma(马鸿洋), and Jun-Qing Liang(梁俊卿)
Chin. Phys. B, 2025, 34 (**3**): 030306. DOI: 10.1088/1674-1056/adab63

Quantum decoder design for subsystem surface code based on multi-head graph attention and edge weighting

Nai-Hua Ji(纪乃华), Hui-Qian Sun(孙汇倩), Bo Xiao(肖博), Ping-Li Song(宋平俐), and Hong-Yang Ma(马鸿洋)
Chin. Phys. B, 2025, 34 (**2**): 020309. DOI: 10.1088/1674-1056/ada1c7

Decoding topological XYZ^2 codes with reinforcement learning based on attention mechanisms

Qing-Hui Chen(陈庆辉), Yu-Xin Ji(姬宇欣), Ke-Han Wang(王柯涵), Hong-Yang Ma(马鸿洋), and Nai-Hua Ji(纪乃华)
Chin. Phys. B, 2024, 33 (**6**): 060314. DOI: 10.1088/1674-1056/ad342b

Recurrent neural network decoding of rotated surface codes based on distributed strategy

Fan Li(李帆), Ao-Qing Li(李熬庆), Qi-Di Gan(甘启迪), and Hong-Yang Ma(马鸿洋)
Chin. Phys. B, 2024, 33 (**4**): 040307. DOI: 10.1088/1674-1056/ad2bef

Performance of entanglement-assisted quantum codes with noisy ebits over asymmetric and memory channels

Ji-Hao Fan(樊继豪), Pei-Wen Xia(夏沛文), Di-Kang Dai(戴迪康), and Yi-Xiao Chen(陈一骁)
Chin. Phys. B, 2023, 32 (**12**): 120304. DOI: 10.1088/1674-1056/acf492

A graph neural network and multi-task learning-based decoding algorithm for enhancing $XZZX$ code stability in biased noise

Bo Xiao(肖博)[1], Zai-Xu Fan(范在旭)[1], Hui-Qian Sun(孙汇倩)[1], Hong-Yang Ma(马鸿洋)[2], and Xing-Kui Fan(范兴奎)[2,†]

[1] *School of Information and Control Engineering, Qingdao University of Technology, Qingdao* 266033, *China*
[2] *School of Sciences, Qingdao University of Technology, Qingdao* 266033, *China*

(Received 25 October 2024; revised manuscript received 6 February 2025; accepted manuscript online 27 February 2025)

Quantum error correction is a technique that enhances a system's ability to combat noise by encoding logical information into additional quantum bits, which plays a key role in building practical quantum computers. The $XZZX$ surface code, with only one stabilizer generator on each face, demonstrates significant application potential under biased noise. However, the existing minimum weight perfect matching (MWPM) algorithm has high computational complexity and lacks flexibility in large-scale systems. Therefore, this paper proposes a decoding method that combines graph neural networks (GNN) with multi-classifiers, the syndrome is transformed into an undirected graph, and the features are aggregated by convolutional layers, providing a more efficient and accurate decoding strategy. In the experiments, we evaluated the performance of the $XZZX$ code under different biased noise conditions (bias = 1, 20, 200) and different code distances (d = 3, 5, 7, 9, 11). The experimental results show that under low bias noise (bias = 1), the GNN decoder achieves a threshold of 0.18386, an improvement of approximately 19.12% compared to the MWPM decoder. Under high bias noise (bias = 200), the GNN decoder reaches a threshold of 0.40542, improving by approximately 20.76%, overcoming the limitations of the conventional decoder. They demonstrate that the GNN decoding method exhibits superior performance and has broad application potential in the error correction of $XZZX$ code.

Keywords: quantum error correction, $XZZX$ code, biased noise, graph neural network

PACS: 03.67.–a, 03.67.Pp, 87.64.Aa

DOI: 10.1088/1674-1056/adbadb **CSTR:** 32038.14.CPB.adbadb

1. Introduction

Quantum computing[1–5] leverages quantum phenomena such as superposition and entanglement, demonstrating the potential to surpass classical computing capabilities. However, the fragility of qubits makes them susceptible to environmental noise such as phase noise, thermal noise, and electromagnetic interference during computation. This leads to quantum state decoherence and measurement errors, resulting in quantum information dissipation. Quantum errror correction (QEC)[6–9] employs redundant encoding to distribute logical qubit information across multiple physical qubits and uses stabilizer measurements and syndrome extraction to detect and correct errors caused by noise. Unlike classical error correction, which typically only needs to handle bit-flip errors, quantum error correction involves more complex error types, including phase-flip errors and bit-phase superposition errors.[10–13]

Since 1995, significant progress has been made in the field of quantum error correction. In 1995, Shor[14] proposed the well-known 9-qubit Shor code. In 1998, Kitaev and Bravyi[15] developed the planar version of the surface code. In 2001, Freedman and Meyer[16] further expanded the concept of the surface code. In 2009, Kolmogorov[17] introduced the Blossom V algorithm, combining the minimum weight perfect matching (MWPM) algorithm with graph theory. In 2020, Andersen[18] implemented the smallest surface code structure using seven superconducting qubits, achieving an average initialization fidelity of 96.1% for encoded logical qubits through high-precision stabilizer measurements. In 2021, Ataides[19] proposed a variant of the surface code, the $XZZX$ code offering new possibilities for fault-tolerant quantum computing. In 2022, Qian Xu[20] explored the performance of the $XZZX$ code under biased noise, revealing that with standard CX gates, the threshold error rate of the $XZZX$ code was 50% higher than that of the surface code. Finally, in 2023, Diego Forlivesi[21] further investigated logical errors in the $XZZX$ code.

In the decoding process of the $XZZX$ code, while the minimum weight perfect matching (MWPM)[22–24] algorithm is widely used, it has notable drawbacks. First, MWPM does not fully utilize the biased noise advantages of the $XZZX$ code, leading to suboptimal performance in asymmetric error environments. Second, it struggles with efficiently handling measurement errors, which are common in fault-tolerant quantum computing. Finally, the computational complexity[25] of MWPM increases significantly with system size, making it

†Corresponding author. E-mail: fanxingkui@126.com

© 2025 Chinese Physical Society and IOP Publishing Ltd. All rights, including for text and data mining, AI training, and similar technologies, are reserved.

less practical for large-scale quantum error correction tasks.

To address these issues, this paper proposes a graph neural network (GNN)-based decoding method for $XZZX$ code.[26,27] Traditional single-task learning methods typically assume that error types are known and fixed. In quantum error correction, especially when multiple noise sources are present, GNN can handle multiple error types simultaneously through multi-task learning, enhancing the model's robustness in noisy environments.In the graph convolutional layer, an attention mechanism [28,29] is introduced to handle the correlation between different nodes. Finally, global feature vectors are generated through a pooling layer and fed into a multi-class classifier layer. The experimental results show that under low noise conditions, the GNN decoder significantly outperforms the conventional MWPM decoder in terms of logic error rate, with a threshold improvement of about 19.12%. Under high noise conditions, the GNN decoder performs even better with a threshold of 0.40542.

The remainder of this paper is organized as follows: In Section 2 we provide background knowledge on the $XZZX$ code and graph neural networks. In Section 3 we describe a detailed description of the proposed decoder architecture and experimental setup. In Section 4 we discuss the error correction performance analysis of the decoder. In Section 5 we discuss the limitations of this study and future research directions.

2. Background

2.1. The $XZZX$ code

In order to evaluate the performance of a quantum code, a noise model[30,31] is typically adopted where each qubit is independently subjected to quantum noise. A general single-qubit Pauli noise channel, expressed as

$$\rho \to (1-p)\rho + p(r_X X\rho X + r_Y Y\rho Y + r_Z Z\rho Z), \tag{1}$$

where $p \in [0,1]$ represents the physical error rate for a single qubit, and the weights r_X, r_Y, and r_Z represent the relative probabilities of Pauli-X, Pauli-Y, and Pauli-Z errors, respectively. For example, in a depolarizing channel, $r_X = r_Y = r_Z = 1/3$, which describes unbiased Pauli noise. Unlike traditional noise models,[32–34] this paper adopts a biased noise model where Z errors are more frequent.

The $XZZX$ code[19–21] demonstrates strong fault-tolerant capabilities in this biased noise environments. Unlike traditional surface codes, each face of the $XZZX$ code has only one stabilizer generator. Specifically, the $XZZX$ code can be defined on the same two-dimensional lattice as the conventional surface code. As shown in Fig. 1(a). Without considering boundaries, the $XZZX$ code has only one type of stabilizer generator. The form of each stabilizer is defined as follows:

$$S_{XZZX} = X \otimes Z \otimes Z \otimes X. \tag{2}$$

However, with boundaries, there are two types of stabilizer generators, the tensor product of Pauli operators XXZ or ZZX. As shown in Fig. 1(b), the $XZZX$ code is characterized by having only one logical error chain unlike the traditional surface code, where X and Z operators alternate along the chain. This structure enables the $XZZX$ code to handle Pauli-Z and Pauli-X errors more effectively by symmetrically treating both types of errors, thus improving its error correction capability.

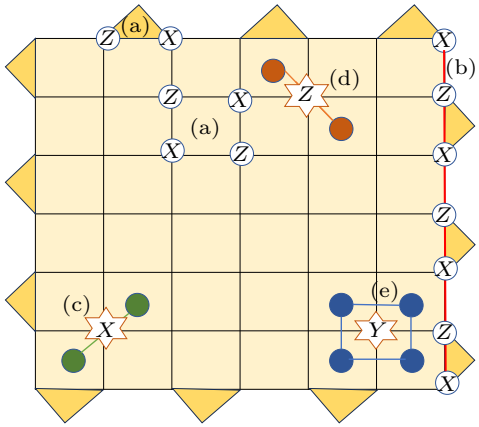


Fig. 1. The $XZZX$ code lattice configuration: (a) the naming of the $XZZX$ code corresponds to its stabilizer operations, with a consistent stabilizer structure on each face; (b) the red line shows the logical error chain on the edges; (c) the green line reprensents a Pauli-X error chain; (d) the orange line reprensents a Pauli-Z error chain; and (e) the blue line reprensents a Pauli-Y error chain.

As shown in Fig. 1(c), when a Pauli-X error occurs, it causes two adjacent defects, and multiple Pauli-X errors form a longer error chain, with these chains aligned in the same direction. Similarly, Pauli-Z errors chains are aligned orthogonally to the Pauli-X error chains, as illustrated Fig. 1(d). Pauli-Y errors result in four defects on adjacent faces like the surface code as shown in Fig. 1(e). By utilizing this characteristic, the $XZZX$ code can dynamically adjust the aspect ratio of the lattice in response to the noise model it encounters, balancing the detection and correction of X and Z errors.

2.2. The graph neural network

Graph Neural Networks (GNN)[35–38] are a class of neural network models specifically designed to handle graph-structured data. In Fig. 2, a graph consists of nodes and edges, denoted as $G = (N, E)$, where an edge $e_{ij} \in E$ represents a connection between two nodes $i, j \in N = \{1, \ldots, m\}$. The graph formed by these nodes serves as the input to the GNN. The core operation of a GNN is to update the features of a node by passing messages between neighboring nodes. In a traditional graph convolutional layer, the updated features of a node are computed based on the aggregation of features from its neighboring nodes, as follows:

$$\boldsymbol{h}_i^{(k+1)} = \sigma\left(\boldsymbol{W}_1 \boldsymbol{h}_i^{(k)} + \sum_{j \in \mathcal{N}(i)} e_{ij} \boldsymbol{W}_2 \boldsymbol{h}_j^{(k)}\right), \tag{3}$$

where $\boldsymbol{h}_i^{(k)}$ represents the feature vector of node i at layer k, $\boldsymbol{W}_1$ and $\boldsymbol{W}_2$ are learnable weight matrices, and σ is an activation function.

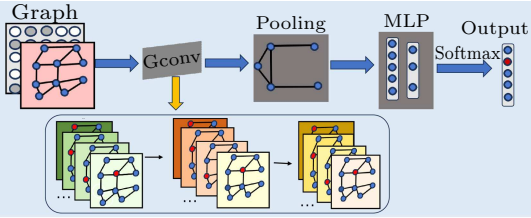


Fig. 2. The process of graph convolutional networks. The input graph first passes through multiple graph convolutional layers to extract features, then global vectors are generated in the pooling layer, and finally processed by a multi-layer network to produce the output.

Additionally, the global pooling layer[39–41] plays a critical role. The basic formula is as follows:

$$h_{\text{pool}} = \frac{1}{N}\sum_{i=1}^{N} h_i^{(L)}, \tag{4}$$

where N represents the number of nodes in the graph, and $h_i^{(L)}$ denotes the output features of the i-th node in the final (L-th) layer. In classification tasks, this feature vector plays a critical role as it integrates the global information of the graph into a high-dimensional vector, which is then passed as input to the classifier. Through this pooling process, the GNN not only captures local node information but also grasps the overall structure of the graph, thereby providing a more comprehensive data representation for downstream classification or prediction tasks.

3. Designing decoding algorithms

3.1. Executing check operators process

The $XZZX$ code is locally equivalent to the traditional CSS surface code, and thus retains its encoding properties. We choose a lattice of size $d_x \times d_z$, using $n = O(d_x d_z)$ qubits, where d_x and d_z represent the minimum lengths of the X and Z logical operators. These logical operators resemble string-like structures that traverse homologically non-trivial paths across opposite boundaries of the code. Under open boundary conditions, the $XZZX$ code encodes one logical qubit. We identify and locate errors in the code by measuring stabilizer operators. Since the code space is defined as the set of $+1$ eigenstates, a measurement result of -1 indicates that the system has deviated from its code space. Our stabilizer measurement[42,43] circuit employs Hadamard gates to transform the standard surface code circuit into an $XZZX$ code circuit.

In Fig. 3, all stabilizer generators can be measured in parallel. The auxiliary qubit is placed at the center of each lattice and initialized in the $|+\rangle$ state. It interacts with two data qubits affected by Pauli-X errors (qubits 2 and 3) through CNOT gates, and with two data qubits affected by Pauli-Z errors (qubits 1 and 4) through CZ gates. Thus, the ancillary qubit collects the parity information of the stabilizers on the lattice. We finally measure the ancillary qubit in the Pauli-X basis to determine the parity result. The stabilizer measurement can be articulated through the expectation value of the stabilizer operator:

$$\langle S \rangle = \langle \psi | Z_1 X_2 X_3 Z_4 | \psi \rangle, \tag{5}$$

where $|\psi\rangle$ represents the quantum state of the system under investigation. An expectation value of $\langle S \rangle = +1$ implies that the state is invariant under the stabilizer operation, confirming that the system resides within the correct logical subspace. Conversely, $\langle S \rangle = -1$ signals a departure from the encoded subspace, indicative of quantum errors affecting one or more qubits.

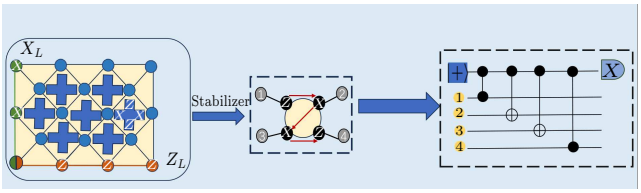


Fig. 3. Stabilizer measurement circuit for the $XZZX$ code. This figure chooses a lattice of size $d_x = 3$, $d_z = 4$ for encoding, where all the checks (except for the three-qubit checks on the boundary) follow the same form. The coupling order of the data qubits and the ancillary qubit (located at the center of each face) is indicated by the arrows. We use the same order for every stabilizer. Before measurement, the four data qubits are coupled to an ancillary qubit, which is initialized in the $|+\rangle$ state and measured in the X basis.

3.2. Mapping algorithm from syndromes to graph

The graph structure mapping of the $XZZX$ code (as shown in Fig. 4) converts the stabilizer measurement results of the qubits into a graph structure that the neural network can process. In the process, each detected error event is represented as a node on the graph,[44–47] and its feature vector is defined as

$$\boldsymbol{X} = (x, y, t), \tag{6}$$

where x and y represent the distances from the nodes on the graph to the north and west boundaries, respectively, while t represents the round of stabilizer measurement.

The experimental design is as follows. First, Pauli-X and Pauli-Z errors are randomly introduced according to the specified noise model. Then, the data qubits and auxiliary qubits are initialized. Next, multiple rounds of stabilizer measurements (rounds = 1, 2, …, n) are performed to extract the syndrome. In each round, we analyze the differences between the current and previous round measurement results to determine if a new error has occurred, and these differences are treated as nodes in the graph. Finally, during the data qubit rounds, the data qubits are measured to record the final quantum state.

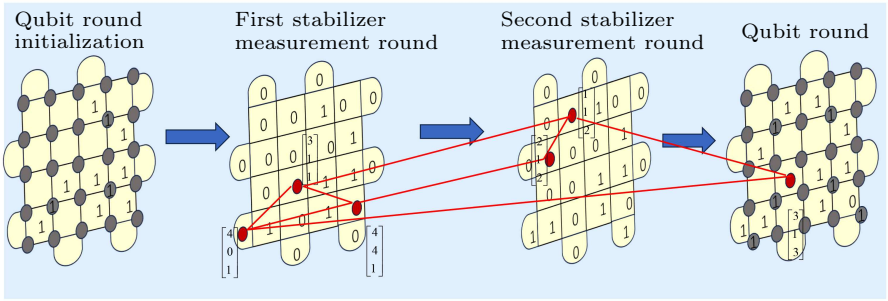


Fig. 4. Quantum error syndrome extraction and graph structure construction process. The figure first configures the stabilizer results based on the initial errors, the numbers on the faces indicate that 0 means no error was detected by the stabilizer, and 1 means an error was detected. Then, two rounds of stabilizer measurements are performed to extract the syndrome. The error information generated by each stabilizer measurement is recorded as a node, and the edges represent possible error propagation paths. The final measurement rounds of the data qubits are also shown.

The graph is constructed as a complete graph where all nodes are initially connected. To optimize computational efficiency, connections between each node and its distant neighbors are removed, only up to three neighbors are retained. This ensures that the number of operations per layer in the neural network scales linearly with the number of detected events, thereby improving computational efficiency. The edge weights are initially set based on the distance between the two connected nodes, represented by the following formula:

$$e_{ij} = \frac{1}{(\max(\|\boldsymbol{p}_i - \boldsymbol{p}_j\|, \varepsilon))^2}, \tag{7}$$

where $\boldsymbol{p}_i$ and $\boldsymbol{p}_j$ are the position vectors of nodes i and j, and ε is a small positive constant to prevent division by zero. By configuring edge weights in this way, the analysis of comprehensive data can be performed more effectively, aiding in the identification of error patterns and providing efficient solutions for quantum error correction, thereby enhancing the network's ability to detect and correct errors.

3.3. Decoding algorithm using graph neural network

In the quantum error correction for the $XZZX$ code, this paper designs a decoder based on a graph neural network (GNN) and introduces a multi-head attention mechanism to handle the complex graph structures, as shown in Fig. 5. The graph structure of the error patterns is constructed based on stabilizer measurement results. In the graph convolutional layers, the GNN updates the node states by gathering features from neighboring nodes and applying weights. However, the correlations between nodes are not equally important. To address this, an attention mechanism[48–51] is introduced to dynamically adjust the edge weights . The formula for updating the node features is as follows:

$$\boldsymbol{h}_i^{(k+1)} = \sigma\left(\boldsymbol{W}_1\boldsymbol{h}_i^{(k)} + \sum_{j\in\mathcal{N}(i)} e_{ij}\alpha_{ij}\boldsymbol{W}_2\boldsymbol{h}_j^{(k)}\right), \tag{8}$$

where $\boldsymbol{h}_i^{(k)}$ represents the feature vector of node i at layer k, $\boldsymbol{W}_1$ and $\boldsymbol{W}_2$ are learnable weight matrices, $N(i)$ is the set of neighbors of node i, e_{ij} is the edge weight between nodes i and j, and α_{ij} is the attention coefficient between nodes i and j. The function σ represents a nonlinear activation function. The attention coefficient α_{ij} is calculated as

$$\alpha_{ij} = \frac{\exp\left(\sigma\left(\boldsymbol{a}^\top[\boldsymbol{W}_1\boldsymbol{h}_i^{(k)}||\boldsymbol{W}_1\boldsymbol{h}_j^{(k)}]\right)\right)}{\sum_{k\in\mathcal{N}(i)}\exp\left(\sigma\left(\boldsymbol{a}^\top[\boldsymbol{W}_1\boldsymbol{h}_i^{(k)}||\boldsymbol{W}_1\boldsymbol{h}_k^{(k)}]\right)\right)}, \tag{9}$$

where $\boldsymbol{a}$ is a learnable attention vector and σ is a nonlinear activation function. To better handle disconnected nodes, the attention coefficients can be adjusted by introducing the adjacency matrix

$$\alpha_{ij} = \mathrm{Softmax}\left(\sigma\left(\boldsymbol{a}^\top[\boldsymbol{W}_1\boldsymbol{h}_i^{(k)}||\boldsymbol{W}_1\boldsymbol{h}_j^{(k)}]\right)\tilde{A}_{ij}\right), \tag{10}$$

where the matrix $\tilde{A}_{ij}$ is the adjacency matrix. In this paper, we use a multi-head attention mechanism, where each attention head independently computes attention weights between nodes and their neighbors. The final node feature is obtained by aggregating the outputs of multiple heads, as expressed by

$$\boldsymbol{h}_i^{(k+1)} = \frac{1}{K}\sum_{K=1}^{K}\sigma\left(\boldsymbol{W}_1^{(k)}\boldsymbol{h}_i^{(k)} + \sum_{j\in\mathcal{N}(i)} e_{ij}\alpha_{ij}^{(k)}\boldsymbol{W}_2^{(k)}\boldsymbol{h}_j^{(k)}\right), \tag{11}$$

where K is the number of attention heads, and $W_1^{(k)}$, $W_2^{(k)}$, and $\alpha_{ij}^{(k)}$ are the weight matrices and attention coefficients for the k-th attention head. By introducing multi-head attention,[41,42] GNN can capture more diverse relationships and patterns within the graph structure. Each attention head independently learns different feature representations, increasing the model's robustness, reducing the bias produced by a single attention head, and ultimately improving the overall performance of the model.

In each GNN layer, multiple hidden channels are used to enhance the model's representation ability. The number of hidden channels gradually increases in each layer, allowing the model to extract more local and global graph structure features. In the final layer of the model, 256 hidden channels and a single attention head are used to generate more definitive prediction results. The multi-head attention mechanism allows each node to learn different neighbor relationship patterns through multiple attention heads, enhancing the model's robustness.

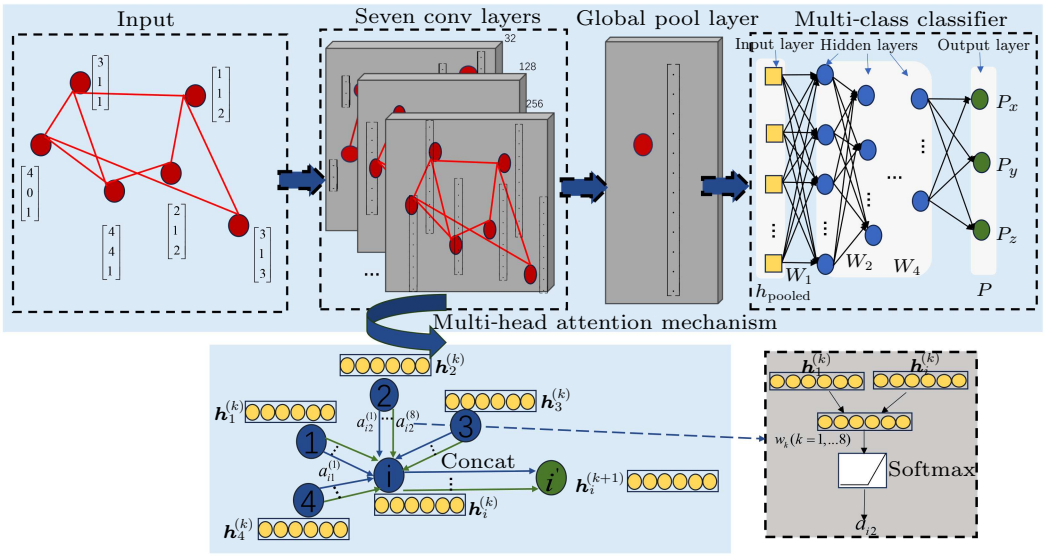


Fig. 5. Graph neural network architecture with multi-head attention mechanism. This diagram illustrates the architecture of a graph neural network (GNN) with a multi-head attention mechanism designed for quantum error correction. The input data are initially mapped onto a graph structure and then processed through multiple convolutional layers. Each convolutional layer extracts different features, and the multi-head attention mechanism enhances the information flow between nodes. A global pooling layer aggregates the node features into a global feature vector, which is then fed into the multi-class classifier.

After several layers of graph convolution, the model uses a global pooling layer to extract the most important node features and aggregates them into a global feature vector for subsequent classification tasks. Then, the model uses a multiclass classifier to compute the probabilities of each type of logical error. Typically, a multiclass classifier uses a Softmax activation function, which converts the raw classification scores into a probability distribution, ensuring that the sum of probabilities for all classes equals 1. The specific formula for this classification process is given by the following equation:

$$P = \text{Softmax}\left(\boldsymbol{W}_{\text{out}}^{\text{final}} \cdot \tanh\left(\alpha \cdot h_{\text{pooled}} + \beta \cdot \sum_{i=1}^{N} W_i h_i^{(L)}\right) + \lambda \cdot ||\boldsymbol{W}_{\text{out}}^{\text{final}}||^2\right), \tag{12}$$

where $\boldsymbol{W}_{\text{out}}^{\text{final}}$ is the final weight matrix, h_{pooled} is the pooled node feature vector, W_i is the weight matrix of node i, $h_i^{(L)}$ is the output feature of node i at layer L, α, β, and λ are hyperparameters, and $||\boldsymbol{W}_{\text{out}}^{\text{final}}||^2$ is the L_2 regularization term. P is the probability distribution of the classification results, representing the probabilities of the Pauli-X, Pauli-Y, and Pauli-Z operations.

To ensure the effectiveness of the model, binary cross-entropy loss is used during training to measure the difference between the predicted results and the true labels:

$$L = -\sum_{c\in\{X,Y,Z\}} (\alpha_c y_c \log(P_c) + \beta_c (1-y_c)\log(1-P_c)) + \lambda\left(\frac{1}{N}\sum_{i=1}^{N}\|\boldsymbol{W}_i\|^2\right), \tag{13}$$

where α_c and β_c are hyperparameters controlling the importance of positive and negative class predictions, y_c is the true label for class c, P_c is the predicted probability for class c, λ is the regularization parameter, N is the total number of nodes, and $\|\boldsymbol{W}_i\|$ represents the L_2 norm of the weight matrix for the i-th node. This formula calculates the cross-entropy loss, measuring the difference between the true labels and the predicted probabilities for each sample. By minimizing this loss, the model learns to predict the correct class (X, Y, or Z) for each sample more accurately. The model weights are updated using the Adam optimizer, dynamically adjusting the learning rate.

3.4. Training process

In the *XZZX* code quantum error correction task, the main software environment is Python 3.9, with model construction and training using the PyTorch Geometric library, running on Ubuntu 20.04 LTS, and utilizing CUDA-accelerated GPUs to enhance training efficiency.

For data generation in the *XZZX* code, different noise bias levels are first set to control the relative frequencies of Pauli-X and Pauli-Z errors. For example, when the bias is 1, Pauli-X and Pauli-Z errors occur with equal probability; when the bias is 200, the frequency of Pauli-Z errors is significantly higher than Pauli-X errors. During the execution of the quantum system, these Pauli-X or Pauli-Z errors are randomly introduced. To detect these errors, multiple rounds of stabilizer measurements are performed. Since errors occur randomly, points that show different results in the next stabilizer measurement compared to the previous round are defined as error nodes. For example, with a bias of 1 and a physical error rate of 0.15, 5000 to 10000 graphs are generated, each containing hundreds of nodes.

Once the graph structure data is generated, the graph neu-

ral network decoder model enters the training phase. First, the generated graph dataset is encapsulated into "Data" objects supported by PyTorch Geometric and loaded in batches for training. During model initialization, multiple layers of graph convolutional networks and multi-head attention mechanisms are used to update node features and capture relationships between adjacent nodes in the graph. During training, the model receives batches of graph data, propagates them through layers, and updates node features, ultimately outputting predictions of logical error probabilities. A binary cross-entropy (BCE) loss function is used to measure the error between the model's predictions and actual labels. The optimization process is managed by the Adam optimizer, which updates the model's weights through backpropagation after each loss calculation to minimize the error. The initial learning rate is set to 0.01, and a learning rate scheduler dynamically adjusts the rate to ensure stable convergence throughout the training process. To prevent overfitting, Dropout is introduced during training. The entire training process is conducted over multiple epochs, and after each epoch, the model is evaluated on the validation set, adjusting the learning rate based on the evaluation results until the model parameters are fully optimized.

4. Results

4.1. Analyzing decoder performance

In the process of quantum error correction, error syndromes are extracted to identify the errors. During the decoding process of the $XZZX$ code, the decoder analyzes all possible error patterns and generates an error model based on the observed syndrome probabilities, outputting the corresponding correction operations. The decoder utilizes the relationship between the total physical error rate P_{total}, the logical error rate P_{logical}, and the code distance d to determine the threshold in order to effectively correct errors. And the threshold is usually defined as the value at the intersection of the physical error rate and the logical error rate curve.

In the experimental design, we simulated different physical error rates and biased noise to evaluate the error-correction performance of multiple decoders on the $XZZX$ code. Simultaneously, each data point was based on different experimental parameters and independently simulated 10^3–10^4 times, with $t = d$ repetitions. This experimental setup covered multiple biased noise conditions (bias = 1, 20, 200) and different code distances (d = 3, 5, 7, 9, 11). The bias noise parameter determined the relative occurrence probability of different types of quantum bit errors, such as the frequencies of Pauli-X and Pauli-Z errors. We used various physical error rates to simulate diverse noise environments. The results showed that the logical error rate of the $XZZX$ code is particularly sensitive to changes in physical error rates under high bias conditions.

As shown in Fig. 6, as the physical error rate increases, the logical error rate also rises. However, under larger code distance conditions, the error correction performance significantly improves, resulting in better suppression of the logical error rate. Specifically, under the condition of bias = 200, before the threshold, a larger code distance demonstrates a clear advantage in reducing the logical error rate. It can be seen that increasing the code distance not only effectively reduces the logical error rate but also enhances the stability and robustness of the decoders under different noise conditions.

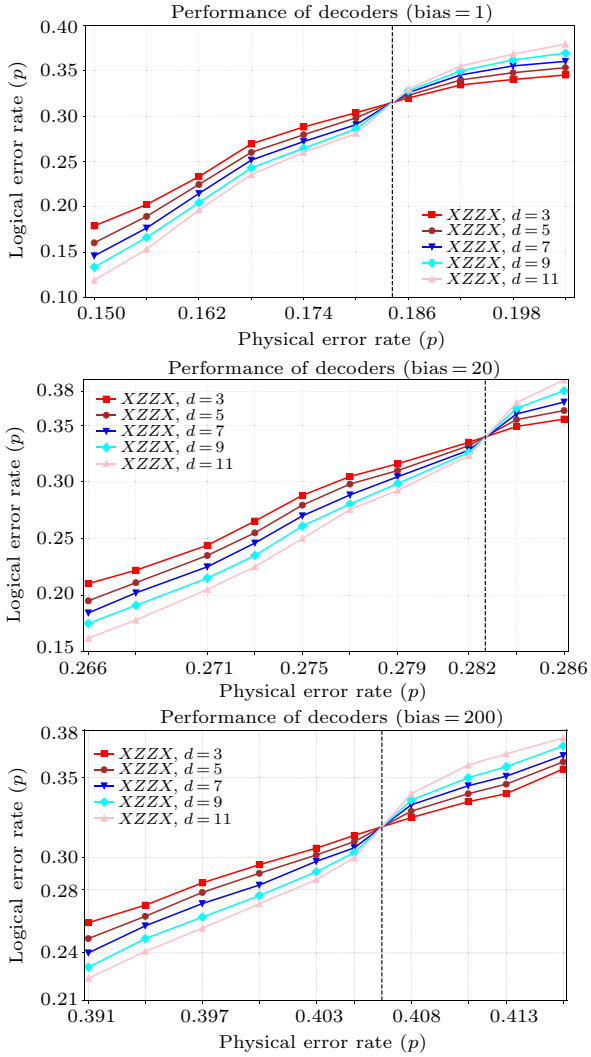


Fig. 6. Logical error rate performance of $XZZX$ code decoders under different bias conditions. This figure illustrates the variation of the logical error rate of $XZZX$ code decoders under different bias conditions (bias = 1, 20, 200) and code distances (d = 3, 5, 7, 9, 11) as the physical error rate increases. The figure shows that as the physical error rate increases, the logical error rate also rises. Under high bias conditions (such as bias = 200), the logical error rate increases more significantly with the rise in physical error rate. However, GAT-based decoders demonstrate better error correction performance, especially under high physical error rates and high bias conditions, effectively suppressing Pauli-Z errors.

To further demonstrate the performance of the decoders, we present Fig. 7, which provides a detailed analysis of the logical error rates of various decoders under different bias conditions. Figure 7 illustrates the logical error rate performance

of multiple decoders (such as GNN and MWPM) under varying bias conditions (bias = 1, 20, 200). The MWPM decoder is based on the minimum weight perfect matching algorithm, which infers errors by analyzing the error chain probabilities between qubits. It uses the probability of each error chain as a distance and infers the most likely error pattern based on these distances to perform error correction.

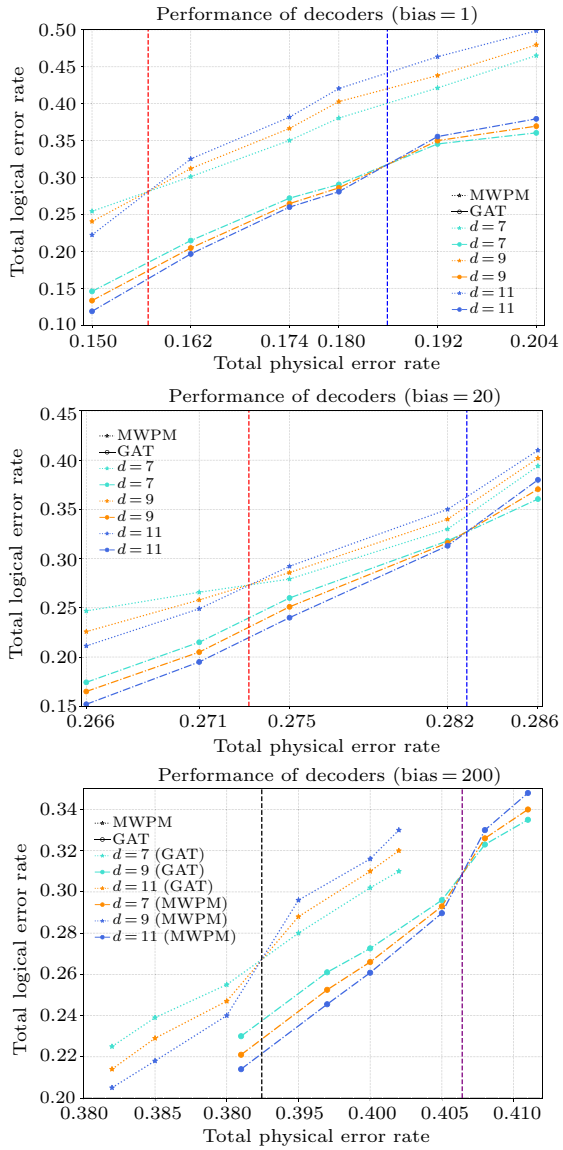


Fig. 7. Performance comparison of decoders for *XZZX* code. This figure illustrates the comparative performance of decoders under three different bias values (bias = 1, 20, 200). Different colors and symbols denote various decoder types and different distance ($d = 7, 9, 11$). The trends in the curves suggest that as the bias increases, the performance of the system varies across different error rates.

In low-bias conditions (such as bias = 1), all decoders exhibit a similar trend in logical error rates at lower physical error rates. However, as the physical error rate increases, the GAT-based decoder gradually demonstrates superior error-correction performance, maintaining a lead in suppressing logical error rates. In high-bias conditions (such as bias = 20 and 200), the logical error rates increase significantly, but the GAT-based decoder continues to perform well, particularly in mitigating Pauli-Z errors, effectively reducing the logical error rate in the high physical error rate region. These results clearly indicate that as the bias conditions intensify, the performance gap between decoders widens, with the GAT decoder standing out prominently in high-bias and high physical error rate environments, proving its significant advantage under biased noise conditions.

4.2. Analysis decoder accuracy

To further explore the accuracy trends across different code distances, figure 8 provides a detailed comparison of the training accuracy and validation accuracy for GNN (graph neural network) decoders with code distances $d = 3, 5, 7, 9, 11$ under bias = 200. From the figure, it can be observed that as the training process progresses, the logical accuracy gradually improves, indicating that the decoder is learning to correct errors more effectively. Across all code distances, the training accuracy steadily increases with more training epochs, but the validation accuracy tends to stabilize in certain cases. This suggests that the model is able to memorize the error patterns present in the training set during the training process. To quantify the performance of the model, accuracy can be calculated as follows:

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total predictions}} \times 100\%, \quad (14)$$

where accuracy represents the proportion of correctly predicted instances to the total number of predictions. This is helpful for evaluating the decoder's performance across different code distances.

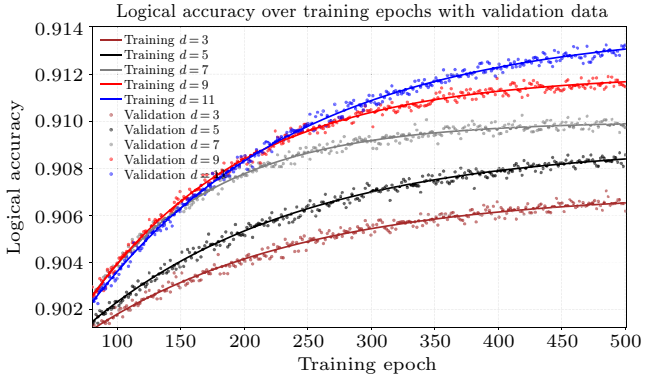


Fig. 8. Training and validation accuracy for different code distances. This figure presents the training and validation accuracy trends for GNN decoders across different code distances ($d = 3, 5, 7, 9, 11$). As the training epochs increase, the logical accuracy improves, demonstrating the decoder's ability to correct errors more effectively. The differences between the training and validation accuracies highlight the model's generalization capability for different code distances.

To gain a deeper understanding of the impact of different model parameters on performance, we added a sensitivity analysis in the experimental section, evaluating the model's performance under different learning rates and layer numbers. Regarding the learning rate, the results showed that with a learning rate of 0.01, the model converged slowly but achieved

stable convergence. As the learning rate increased, the convergence speed improved, but some instability occurred during training. As for the number of layers, increasing the layers from 4 to 6 improved the accuracy, but further increasing the layers resulted in diminishing returns, with signs of overfitting.

5. Conclusion

We proposed an advanced decoding framework based on the *XZZX* quantum error-correcting code, combined with graph neural networks (GNN) and attention mechanisms. Through this approach, the GNN can adaptively learn error patterns, enabling the decoder to achieve better robustness and accuracy when facing different types of quantum errors. Furthermore, the introduction of the attention mechanism allows the model to dynamically adjust the importance of features, optimizing the decoding process. This flexibility is particularly crucial in addressing complex noise conditions.

Future research directions can focus on several key areas to further enhance the performance of the quantum error correction decoding framework based on graph neural networks (GNN). First, integrating temporal graph neural networks could be considered to capture the temporal dynamics of quantum errors. Additionally, exploring the potential of combining this framework with other machine learning techniques, such as reinforcement learning, could assist the decoder in optimizing its decoding strategies based on real-time feedback, thereby improving performance in high-noise environments. Furthermore, applying this framework to other types of quantum error correction codes, such as low-density parity-check (LDPC) codes, may expand its applicability and enhance its versatility.

Acknowledgements

Project supported by the Natural Science Foundation of Shandong Province, China (Grant No. ZR2021MF049), the Joint Fund of Natural Science Foundation of Shandong Province, China (Grant Nos. ZR2022LL.Z012 and ZR2021LLZ001), and the Key Research and Development Program of Shandong Province, China (Grant No. 2023CXGC010901).

References

[1] Steane A 1998 *Rep. Prog. Phys.* **61** 117
[2] Gyongyosi L and Imre S 2019 *Comput. Sci. Rev.* **31** 51
[3] Herman D, Googin C and Liu X 2023 *Nat. Rev. Phys.* **5** 450
[4] Hassija V, Chamola V and Saxena V 2020 *ET Quantum Commun.* **1** 42
[5] Huang HL, Wu D and Fan D 2020 *Sci. China Inf. Sci.* **63** 180501
[6] Wang H W, Xue Y J, Ma Y L, Hua N and Ma H Y 2022 *Chin. Phys. B* **31** 010303
[7] Chen Q H, Ji Y X, Wang K H, Ma H Y and Ji N H 2024 *Chin. Phys. B* **33** 060314
[8] Chiaverini J, Leibfried D and Schaetz T 2004 *Chin. Phys. B* **33** 040307
[9] Aoki T, Takahashi G and Kajiya T 2009 *Nat. Phys.* **5** 541
[10] Gong L H, Li M L, Cao H and Wang B 2024 *Phys. Rev. B* **21** 055209
[11] Chao R and Reichardt B W 2018 *Phys. Rev. Lett.* **121** 050502
[12] Krinner S, Lacroix N and Remm A 2022 *Nature* **605** 669
[13] Beale S J, Wallman J J and Gutiérrez M 2018 *Phys. Rev. Lett.* **121** 190501
[14] Shor P W 1995 *Phys. Rev. A* **52** R2493
[15] Bravyi S and Kitaev A Y 1998 arXiv: 9811052v1 [quant-ph]
[16] Freedman M H and Meyer D A 2001 *Found. Comput. Math.* **1** 325
[17] Kolmogorov V 2009 *Math. Prog. Comp.* **1** 43
[18] Andersen C K, Remm A and Lazar S 2020 *Nat. Phys.* **16** 875
[19] Bonilla Ataides J P, Tuckett D K and Bartlett S D 2021 *Nat. Commun.* **12** 2172
[20] Xu Q, Mannucci N and Seif A 2022 arXiv: 2203.16486 [quant-ph]
[21] Forlivesi D, Valentini L and Chiani M 2023 *IEEE J. Sel. Areas Commun.* **42** 1808
[22] Fowler A G, Whiteside A C and Hollenberg L C L 2012 *Phys. Rev. A* **86** 042317
[23] Kam JF, Gicev S, Modi K, Southwell A and Usman M 2024 arXiv: 2410.23779 [quant-ph]
[24] Chatterjee A, Das S and Ghosh S 2023 arXiv: 2308.02769 [quant-ph]
[25] Kuo K Y and Lai C Y 2023 *IEEE Int. Symp. Inf. Theory (ISIT)* June 26–July 1, 2022, Espoo, Finland, pp. 186–191
[26] Zheng J, Gao Q and Lü Y 2021 *Proc. 40th Chinese Control Conf. (CCC)* (Shenyang, China: IEEE) pp. 6335–6340
[27] Lange M, Havstrm P and Srivastava B 2023 arXiv: 2307.01241 [quant-ph]
[28] Niu Z, Zhong G and Yu H 2023 *Neurocomputing* **452** 48
[29] Guo M H, Xu T X and Liu J J 2023 *Comp. Visual Media* **8**331
[30] Resch S and Karpuzcu U R 2021 *ACM Computing Surveys* **54** 1
[31] Du Y, Hsieh M H, Liu T and Tao D 2020 arXiv: 2003.09416 [quant-ph]
[32] Higgott O, Bohdanowicz T C, Kubica A and Flammia S T 2023 *Phys. Rev. X* **13** 031007
[33] Harper R, Flammia S T and Wallman J J 2020 *Nat. Phys.* **16** 1184
[34] Sung Y and Braumüller J 2021 *Nat. Comm.* **12** 967
[35] Wu Z, Pan S and Chen F 2020 *IEEE Trans. Neural Netw. Learn. Syst.* **32** 4
[36] Asif N A, Sarker Y and Chakrabortty R K 2021 *IEEE Access* **9** 60588
[37] Velickovic P 2023 *Curr. Opin. Struct. Biol.* **79** 102538
[38] Luan S, Hua C and Lu Q 2022 *Adv. Neural Inf. Process. Syst.* **35**1362
[39] Nirthika R, Manivannan S and Ramanan A 2022 *Neural Comput. Appl.* **34** 5321
[40] Özdemir C 2023 *Expert Syst. Appl.* **223** 119892
[41] Kurdija A S and Vladimir K 2022 *IEEE Access* **10** 10402
[42] Lavasani A, Alavirad Y and Barkeshli M 2021 *Nat. Phys.* **17**342
[43] Krinner S, Lacroix N and Remm A 2022 *Nature* **605** 669
[44] Jiang W and Luo J. 2022 *Expert Syst. Appl.* **207** 117921
[45] Kerstin B 2022 arXiv: 2205.08154 [quant-ph]
[46] Marcianò A, Terzidis K and Zappal E 2022 *Expert Syst. Appl.* **153** 164
[47] Varsamopoulos S, Bertels K and Almudever C G 2019 *IEEE Trans. Comput.* **69** 300
[48] Wen Z, Lin W and Wang T 2023 *Biomimetics* **8** 199
[49] Sharaf Al-deen H S, Zeng Z and Al-sabri R 2021 *Appl. Syst. Innov.* **4** 85
[50] Chen Y, Zhuang T and Guo K 2021 *Appl. Intell.* **51** 4287
[51] Tiku S, Gufran D and Pasricha S 2022 *IEEE Int. Conf. Indoor Positioning and Indoor Navigation (IPIN)*, Beijing, China, pp. 1–8