

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)

Citation : Chin. Phys. B, 2025, 34 (**2**): 020309. DOI: 10.1088/1674-1056/ada1c7
Journal homepage: http://cpb.iphy.ac.cn; http://iopscience.iop.org/cpb

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

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

Approximate error correction scheme for three-dimensional surface codes based reinforcement learning

Ying-Jie Qu(曲英杰), Zhao Chen(陈钊), Wei-Jie Wang(王伟杰), and Hong-Yang Ma(马鸿洋)
Chin. Phys. B, 2023, 32 (**10**): 100307. DOI: 10.1088/1674-1056/acd8a9

An overview of quantum error mitigation formulas

Dayue Qin(秦大粤), Xiaosi Xu(徐晓思), and Ying Li(李颖)
Chin. Phys. B, 2022, 31 (**9**): 090306. DOI: 10.1088/1674-1056/ac7b1e

---

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

Nai-Hua Ji(纪乃华)[1,†], Hui-Qian Sun(孙汇倩)[1], Bo Xiao(肖博)[1], Ping-Li Song(宋平俐)[1], and Hong-Yang Ma(马鸿洋)[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 16 December 2024; accepted manuscript online 20 December 2024)

Quantum error-correcting codes are essential for fault-tolerant quantum computing, as they effectively detect and correct noise-induced errors by distributing information across multiple physical qubits. The subsystem surface code with three-qubit check operators demonstrates significant application potential due to its simplified measurement operations and low logical error rates. However, the existing minimum-weight perfect matching (MWPM) algorithm exhibits high computational complexity and lacks flexibility in large-scale systems. Therefore, this paper proposes a decoder based on a graph attention network (GAT), representing error syndromes as undirected graphs with edge weights, and employing a multi-head attention mechanism to efficiently aggregate node features and enable parallel computation. Compared to MWPM, the GAT decoder exhibits linear growth in computational complexity, adapts to different quantum code structures, and demonstrates stronger robustness under high physical error rates. The experimental results demonstrate that the proposed decoder achieves an overall accuracy of 89.95% under various small code lattice sizes ($L = 2, 3, 4, 5$), with the logical error rate threshold increasing to 0.0078, representing an improvement of approximately 13.04% compared to the MWPM decoder. This result significantly outperforms traditional methods, showcasing superior performance under small code lattice sizes and providing a more efficient decoding solution for large-scale quantum error correction.

Keywords: quantum error correction, graph attention network, subsystem surface code, circuit-level noise

PACS: 03.67.–a, 87.64.Aa, 03.67.Pp

DOI: 10.1088/1674-1056/ada1c7 **CSTR:** 32038.14.CPB.ada1c7

1. Introduction

As quantum computing advances, the sensitivity of qubits to environmental noise and errors has become a major bottleneck for large-scale quantum computing (Refs. [1–4]). The collapse of the wave function during measurement leads to irreversible loss of quantum state information, rendering traditional error correction methods ineffective in the quantum domain. Therefore, quantum error correction codes[5] (QECC) have become crucial for ensuring the stability of quantum computing. QECC distributes the information of logical qubits across multiple physical qubits, allowing the recovery of original data even when some qubits are affected by noise. In fault-tolerant quantum computing, QECC not only corrects errors but also maintains the continuity and accuracy of computation in high-noise environments, ensuring stable system operation.[6]

However, as the complexity of quantum computing architectures increases, the demands on quantum error correction codes have become increasingly stringent. In 1997, Kitaev proposed the surface code,[7] which became a core quantum error correction scheme due to its topological protection, high noise resistance, and scalability. However, its measurement process is complex, and in large-scale systems, it can easily increase operational error rates. To address this issue, Bravyi *et al.* proposed the subsystem surface code (SSC)[8] with three-qubit check operators in 2013. By using XXX and ZZZ operations, they simplified the four-qubit checks to three-qubit measurements, reducing implementation difficulty and measurement complexity while retaining the advantages of topological protection. In recent years, SSC has been widely applied to superconducting qubits and ion trap platforms (Refs. [9–11]), reducing the coupling requirements between qubits, improving error correction efficiency, and enhancing hardware compatibility, demonstrating promising application potential.

Despite the advantages of SSC, existing decoding methods face challenges. The current SSC decoding relies mainly on the minimum weight perfect matching (MWPM) algorithm,[8] but it has limitations: the computational resource demand grows exponentially with code distance, reducing correction efficiency; MWPM depends on fixed codeword shapes, making it difficult to adapt to different quantum code structures; and its global search nature significantly increases computational complexity in large-scale quantum systems, limiting practical use (Refs. [12–15]).

To address these challenges, this paper proposes a de-

†Corresponding author. E-mail: 13964863452@126.com
‡Corresponding author. E-mail: hongyang_ma@aliyun.com

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

coder based on the graph attention network (GAT) (Refs. [16–18]) for decoding SSC. Compared to traditional graph neural networks (GNN) (Refs. [19–21]), the GAT offers greater flexibility and expressive power due to its attention mechanism. The decoder transforms the error syndrome data into an undirected graph with edge weights and uses the attention mechanism to dynamically adjust node weights while leveraging GPU parallel processing to accelerate decoding.[22] The computational complexity of GAT grows linearly with the number of nodes and edges, and although the multi-head attention mechanism slightly increases complexity, it remains within a manageable range. Experimental results show that under circuit-level noise, the GAT decoder outperforms MWPM in terms of logical error rate and accuracy across different code lattice sizes ($L = 2, 3, 4, 5$) and error models, particularly demonstrating stronger robustness under high physical error rates, with the error threshold increased to 0.0078.

The main content of this paper is as follows: Section 2 introduces the basic concepts of the subsystem surface code and graph neural networks; Section 3 discusses the input and overall architecture of the GAT decoder; Section 4 evaluates the performance of the decoder; and Section 5 summarizes the research findings and proposes future research directions.

2. Background and related works

2.1. Subsystem surface code

In the field of quantum error correction, topological quantum codes (Refs. [23–26]) utilize global topological properties to identify and correct local errors through non-local features. The surface code is a typical example, arranging qubits on a two-dimensional (2D) lattice and detecting errors using stabilizer operators (Refs. [27–29]). The subsystem surface code (SSC)[8] is a variant of the surface code, which simplifies operational complexity by measuring the parity of XXX and ZZZ Pauli operators on three qubits.

In the present work, we encode a logical qubit using the SSC under open boundary conditions. The topological structure of the SSC includes rough and smooth boundaries, defined on an $L \times L$ square lattice, containing a total of $3L^2 + 4L + 1$ qubits. The stabilizers of the SSC consist of internal weight-6 $X^{\otimes 6}$ and $Z^{\otimes 6}$ stabilizers, as well as boundary weight-2 XX and ZZ stabilizers. Each lattice cell contains four weight-3 triangular gauge generators, and errors are detected by measuring the qubits (black dots) as shown in Fig. 1. The X-type gauge generators are oriented in the NE and SW directions, labeled as $\mathcal{A}$ and $\mathcal{A}'$, while the Z-type generators are oriented in the NW and SE directions, labeled as $\mathcal{B}$ and $\mathcal{B}'$. The weight-6 stabilizers are formed by combining two same-type gauge generators within the lattice, i.e., $\lambda(X^{\otimes 6} = \mathcal{A}\mathcal{A}')$ and $\lambda(Z^{\otimes 6} = \mathcal{B}\mathcal{B}')$, and errors are detected through virtual qubits (white dots) to determine the error condition.

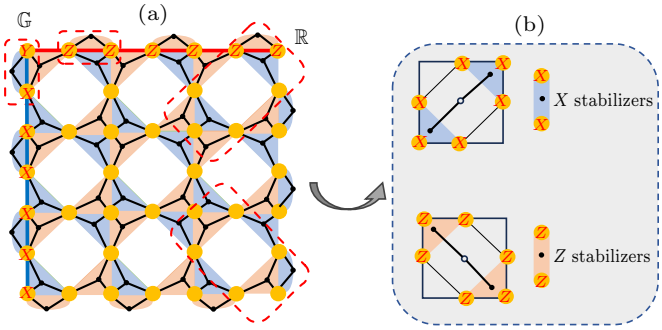


Fig. 1. Structures of the subsystem surface code. In panel (a), yellow dots represent qubits, and blue and orange triangles represent X and Z gauge generators, respectively. The red dashed box marks four different types of stabilizers. The horizontal red line on the boundary indicates a logical Z error, and the vertical green line indicates a logical X error. Panel (b) displays the detailed configuration of the stabilizers, with white dots representing virtual qubits used for stabilizer measurement. The measurement outcomes of the two weight-2 stabilizers are directly provided by the measurement qubits.

In SSC, the logical operators (Refs. [30–32]) commute with the stabilizers but are not part of the stabilizers, allowing logical operations to be performed directly within the encoded space without the need for stabilizer measurements to monitor state changes. The logical operators X_{L} and Z_{L}, as shown by $\mathbb{G}$ and $\mathbb{R}$ in Fig. 1, are defined by the sets of qubits along the horizontal and vertical directions, respectively, and can be expressed as

$$X_{\mathrm{L}}^1 = \prod_{i\in\mathbb{R}} X_i,\ Z_{\mathrm{L}}^1 = \prod_{i\in\mathbb{G}} Z_i,\ X_{\mathrm{L}}^2 = \prod_{i\in\mathbb{G}} X_i,\ Z_{\mathrm{L}}^2 = \prod_{i\in\mathbb{R}} Z_i. \quad (1)$$

The logical operators on the boundary of the SSC only include the horizontal X_{L} and the vertical Z_{L} due to the presence of weight-2 stabilizers on the boundary. The code distance d is defined as the minimum length of an error chain required to perform a non-trivial logical operation[33] and is typically proportional to the lattice size L, i.e., $d = L$,[8] which determines the system's error-correcting capability.

2.2. Graph neural networks

Graph neural network (GNN) (Refs. [34–37]) is a deep learning architecture designed to handle graph-structured data. Unlike convolutional neural networks (CNN) (Refs. [38–40]) and recurrent neural networks (RNN) (Refs. [41–43]), GNN can process data with dynamic topology. The input is the structure of the graph, represented as $G = (V, E)$, where $i, j \in V = \{1, 2, \ldots, m\}$ represent the set of nodes, and $e_{ij} \in E$ represents the set of edges. Each node has a feature vector $\boldsymbol{X}$ with a dimension of $\mathbb{B}^n$, where n is the number of features. The main components of GNN are the graph convolutional layer and the pooling layer.

The graph convolutional layer is the core of GNN, updating node representations by integrating features from neighboring nodes. It operates using the adjacency matrix and feature matrix, often combined with a nonlinear activation function. The corresponding formula can be found in the PyTorch

Geometric library:

$$\boldsymbol{X}_i^{l+1} = \sigma\left(\sum_{j\in\mathcal{N}(i)} e_{ij}\boldsymbol{M}^{(l)}\boldsymbol{X}_j^{(l)} + \boldsymbol{M}_1{}^{(l)}\boldsymbol{X}_i^{(l)} \right), \tag{2}$$

where $\boldsymbol{X}_j{}^{(l)}$ is the feature vector of node j at layer l, $\mathcal{N}(i)$ represents the set of neighboring nodes of node i, $\boldsymbol{M}$ is the weight matrix, σ is the nonlinear activation function.

The pooling layer is used to simplify the graph structure in GNN by reducing the number of nodes, abstracting the graph while retaining key features. Mean pooling is a common method (Refs. [44,45]), the formula is

$$\boldsymbol{X}^{(l+1)} = \frac{1}{m}\sum_{i=1}^{m}\boldsymbol{X}_i^{(l)}, \tag{3}$$

where m represents the number of nodes in the input graph.

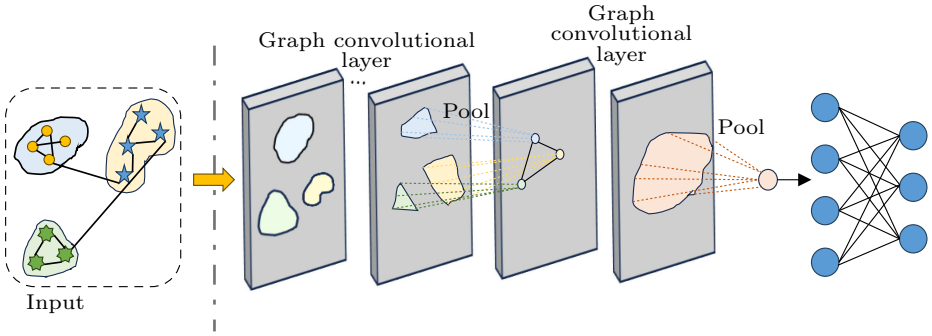


Fig. 2. The "input" on the left represents the input data, where each node and edge represents the structure of the graph. The input graph passes through a graph convolutional layer, which aggregates the features of neighboring nodes to update the current node's features. Then, pooling operations are applied to reduce computational complexity and extract global information. Finally, the processed graph is passed through a fully connected layer for task prediction.

3. Graph attention network decoder model design

3.1. Extracting syndrome process

When performing quantum error correction using SSC, the most critical step is syndrome extraction. For stabilizers located at the boundaries, it is sufficient to determine the eigenvalue of the measurement operator. However, for weight-6 stabilizers, the measurement process is more complex and must be conducted in two steps.The first step involves measuring and recording the eigenvalues of the two triangular gauge generators corresponding to the stabilizer. To measure a single gauge generator, we implement it using a quantum circuit, as shown in Fig. 3(b). This circuit employs three CNOT gates and an ancillary qubit to extract the eigenvalue of the gauge generator over five measurement rounds. The eigenvalue can be expressed as $\lambda(J) = \pm 1$, where $J \in (\mathcal{A}, \mathcal{A}', \mathcal{B}, \mathcal{B}')$, with $\lambda(J) = -1$ indicating the presence of an error. This measurement step lays the groundwork for the subsequent error correction process.

The second step requires the introduction of a virtual qubit, which is controlled by the eigenvalues of the two diagonal gauge generators in the lattice. The specific formula is as follows:

$$\lambda(X^{\otimes 6}) = \lambda(\mathcal{A})\lambda(\mathcal{A}'), \tag{4}$$

$$\lambda(Z^{\otimes 6}) = \lambda(\mathcal{B})\lambda(\mathcal{B}'). \tag{5}$$

A defect is generated on the stabilizer only when the error anticommutes with the stabilizer. For the virtual qubits we assess, if it is an X stabilizer, the measurement result is placed at the position of gauge generator $\mathcal{A}$; otherwise, it is placed at the position of gauge generator $\mathcal{B}$. Figure 3 illustrates the detailed process of measurement and syndrome extraction.

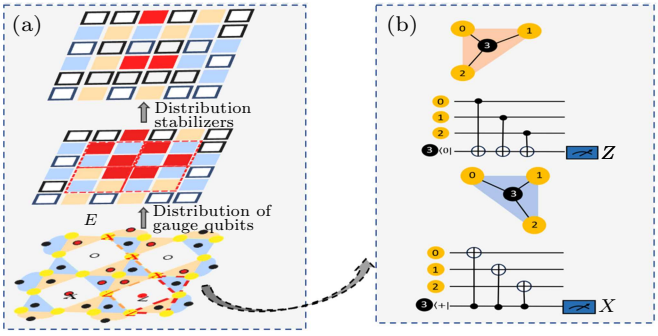


Fig. 3. Panel (a) shows the syndrome extraction process of the subsystem surface code. At the bottom layer, E represents the X error chain, and the red circles indicate the measurement qubits and virtual qubits where errors have occurred. The middle diagram shows the measurement positions of the gauge generators and the error locations, with the dashed box highlighting the area used to determine stabilizer errors. The top diagram illustrates the types of stabilizers and the error locations. Panel (b) shows the measurement circuits for the ZZZ and XXX gauge generators. The yellow circles represent data qubits, and the black circles represent measurement qubits, with black numbers indicating the measurement sequence. The circuit on the top is used to measure the ZZZ gauge generator, with the measurement qubit initialized in the $\langle 0|$ state, while the circuit on the bottom is used to measure the XXX gauge generator, with the measurement qubit initialized in the $\langle +|$ state. Both circuits contain three CNOT gates.

In quantum error correction, when the probability of error for each qubit is independent and equal to p, the goal is to find a recovery operator E' corresponding to the error operator E, in order to restore the quantum state from the error state back to the code space. This can be achieved by using the syndrome as input to the decoder. Through a series of decoding processes, E' can be determined, allowing the quantum state

to be recovered from the error state to the correct state in the code space.

3.2. Graphing syndromes and extracting edge weights process

To input syndrome data into the GAT, it first needs to be converted into the structure of an undirected graph, as shown in Fig. 4. The black dots represent detection events, indicating stabilizer positions that differ from the previous round of measurements. Each detection event corresponds to a vertex in the graph, and each vertex is connected to all other vertices in the graph, forming a complete graph. In this complete graph, each vertex contains five features, which can be represented by the vector $\boldsymbol{X} = (a, b, x, y, t)$. Here, (a, b) are mutually exclusive binary labels (using one-hot encoding), where $(1, 0)$ represents an X-type stabilizer, and $(0, 1)$ represents a Z-type stabilizer; (x, y, t) represent the spatiotemporal coordinates of the vertex. The edge weights in the graph are calculated based on the temporal and spatial Euclidean distances between vertices, thereby indicating the relative importance of each edge. The formula is as follows:

$$e_{ij} = \left(\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2 + (t_i - t_j)^2}\right)^{-2}, \tag{6}$$

where i, j represent the set of detector events, $0 \le x, y \le d$, $1 \le t \le d_t \pm 1$. When only X errors occur, the X stabilizers undergo $d_t + 1$ measurement rounds, while the Z stabilizers undergo $d_t - 1$ measurement rounds. By introducing edge weights, the influence between errors is clearly depicted, facilitating the pruning of the graph. Each vertex retains only its heaviest edges until a preset maximum node degree of 5 is reached, thereby reducing the graph complexity and improving computational efficiency. This results in an undirected graph and corresponding data structure that are input into the graph neural network.

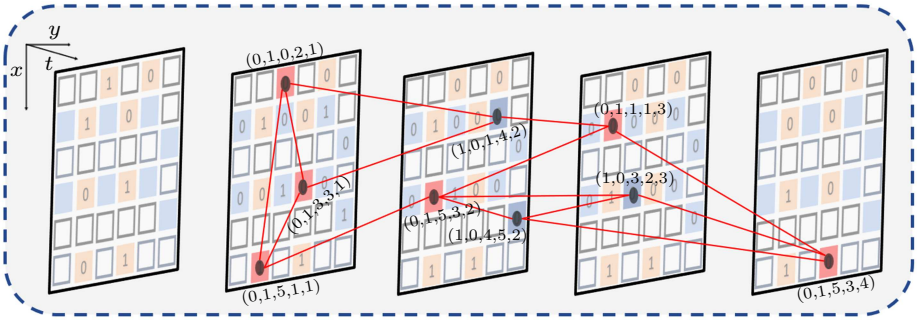


Fig. 4. After $d_t = 3$ measurement rounds, the syndrome is extracted based on the memory experiment design of the subsystem surface code, and the process of generating an undirected graph is depicted. The orange squares represent Z stabilizers, the blue squares represent X stabilizers, and the black dots indicate stabilizer positions where the measurement results differ from the previous round, i.e., error locations. The red edges connect two errors, and the edge weights are calculated based on the positions of these errors. The first layer represents the syndrome corresponding to the initialized data qubits, the middle three layers represent the syndromes from the stabilizer measurement rounds, and the last layer represents the syndrome measured on the data qubits.

The generation of the above undirected graph relies on the memory experiment design based on the subsystem surface code. At the beginning of the experiment, the quantum system is initialized to a known initial state. To effectively reduce measurement errors caused by noise, especially circuit-level noise, multiple rounds of stabilizer measurements are introduced, with each round recording the corresponding syndrome information. After the multiple measurement rounds, the final logical state is measured and compared with the initial state to determine logical errors. The final result is represented by the logical labels (λ_X, λ_Z), with the specific formulas as follows:

$$\lambda_Z = \frac{1}{2}|\langle U_Z\rangle_{t=0} - \langle U_Z\rangle_{t=d_t+1}|, \tag{7}$$

$$\lambda_X = \frac{1}{2}|\langle U_X\rangle_{t=0} - \langle U_X\rangle_{t=d_t+1}|, \tag{8}$$

where $(\lambda_X, \lambda_Z) \in \{0, 1\}$, indicating whether an X or Z logical error has occurred. When $\lambda = 1$, it means a logical error has occurred; otherwise, no logical error has occurred. $\langle U_Z\rangle_{t=0}$ and $\langle U_X\rangle_{t=0}$ represent the measurement results of the logical Z or X stabilizers at the initial moment $t = 0$, while $\langle U_Z\rangle_{t=d_t+1}$ and $\langle U_X\rangle_{t=d_t+1}$ represent the measurement results of the logical Z or X stabilizers at the final measurement round $t = d_t + 1$.

In physical experiments, due to the non-commutativity of U_Z and U_X, only one type of logical error can be detected in each experiment, meaning that each data point corresponds to a single label (λ_X or λ_Z). For example, in a memory-X experiment, the logical label λ_X indicates whether a logical bit has flipped, and this label is determined by measuring the difference between the initial and final states of U_X. However, it is important to note that each data point is assigned only one label, and the two labels (λ_X and λ_Z) never appear simultaneously. Therefore, although the detectors provide information about both labels (λ_X and λ_Z), each data point is marked with only one of them. This leads to the training data being partially labeled.

Since each data point has only one label and these labels

do not fully match all the available information, the neural network decoder must learn to effectively decode based on the partially labeled data. In this setting, the decoder not only needs to learn how to predict λ_X or λ_Z, but also to extract useful information from the unlabeled detector data. Thus, this work adopts a semi-supervised learning framework, leveraging both the partially labeled data and a large amount of unlabeled data to improve the decoder's performance and accuracy in practical applications.

3.3. Designing multi-head graph attention mechanism process

After converting the error syndrome into an undirected graph, the next step is to input it into the GNN for processing. However, while traditional GNN are effective at handling relationships between nodes, they can face high computational complexity, particularly when dealing with large graph structures. The high computational complexity of traditional GNN arises because, in each layer, GNN need to perform weighted averaging or aggregation over all neighboring nodes, and the computational load typically scales with the size of the graph (the number of nodes and edges). Additionally, when updating node features, GNN assign the same weight to all neighboring nodes, which may ignore the differences in relationships between neighbors, thus failing to accurately capture the complex local structural information within the graph. As the graph size increases, the computational complexity and memory requirements of traditional GNN grow significantly, posing challenges to the efficiency when processing large-scale graph structures.

In contrast, convolutional neural networks (CNN), while excelling at processing image-like regular grid structures, are not suitable for graph data, which is a non-Euclidean structure. Specifically, CNN assume the data is structured in a regular grid, and the relationships between nodes are fixed and local. This assumption makes CNN limited in handling graph data, where connections between nodes are dynamic and irregular. For tasks like quantum error correction, CNN have strict requirements on the codeword structure, and are not a general-purpose decoder. Thus, CNN are unable to flexibly adapt to different graph structures and are ineffective at capturing the complex and varying relationships between nodes in a graph.

Therefore, to more flexibly handle the structural properties of graph data and efficiently capture the complex relationships between nodes, we introduce the GAT as the core architecture of the decoder. Compared to traditional GNN, GAT incorporates an attention mechanism, allowing each node to dynamically calculate the weights of its neighboring nodes when updating its features. This mechanism effectively reduces the computation of less important neighbors, thereby lowering the overall computational complexity, especially when dealing with large-scale graphs. Specifically, when updating node features, GAT no longer assigns the same weight to all neighbors but adapts the weights based on the relative importance of each neighbor, ensuring that each node only focuses on the most relevant neighbors and thus reducing the computation on irrelevant nodes. As an enhanced version of GNN, GAT dynamically calculates the weights between nodes, providing a more accurate representation of the complex relationships within the graph structure, and effectively improving the efficiency of the decoder when handling complex graph structures.

The GAT dynamically adjusts the influence of neighboring nodes on the target node during the feature update process using an attention mechanism. It is an improvement based on the graph attention convolutional layer from the PyTorch geometric library. The formula for the graph attention convolutional layer is as follows:

$$\boldsymbol{X}_i^{l+1} = \sigma\left(\alpha_{ij}\left(\sum_{j\in\mathcal{N}(i)} \boldsymbol{M}^{(l)}\boldsymbol{X}_j{}^{(l)} + b^{(l)}\right)\right), \tag{9}$$

where, α_{ij} is the attention coefficient, representing the influence of node j on node i. It allows the model to focus on the neighboring nodes that are more relevant to the current node when updating the node's features. The specific working principle is as follows: first, an unnormalized attention score α'_{ij} is computed for the connected nodes j and i

$$\begin{aligned}\alpha'_{ij} = \ & \text{LeakyReLU}(\boldsymbol{a}^T[(\boldsymbol{M}^{(l)}\boldsymbol{X}_i^{(l)} \\ & + b^{(l)})\|(\boldsymbol{M}^{(l)}\boldsymbol{X}_j{}^{(l)} + b^{(l)})]).\end{aligned} \tag{10}$$

Then, the unnormalized attention score α'_{ij} is normalized into the attention weight α_{ij} using the softmax function

$$\alpha_{ij} = \frac{\exp(\alpha'_{ij})}{\sum_{k\in\mathcal{N}(i)}\exp(\alpha'_{ik})}, \tag{11}$$

where, $\boldsymbol{a}$ is a learnable weight vector, and $(\boldsymbol{M}^{(l)}\boldsymbol{X}_i^{(l)} + b^{(l)})\|(\boldsymbol{M}^{(l)}\boldsymbol{X}_j{}^{(l)} + b^{(l)})$ represents the concatenation of $\boldsymbol{M}^{(l)}\boldsymbol{X}_i^{(l)} + b^{(l)}$ and $\boldsymbol{M}^{(l)}\boldsymbol{X}_j{}^{(l)} + b^{(l)}$.

Building on this, this paper introduces the calculation of edge weights, allowing the model to not only perform weighted aggregation using the attention weights of neighboring nodes but also to further adjust the features of neighboring nodes using the edge weight e_{ij}. That is

$$\boldsymbol{X}_i^{l+1} = \sigma\left(\alpha_{ij}\left(\sum_{j\in\mathcal{N}(i)} e_{ij}\boldsymbol{M}^{(l)}\boldsymbol{X}_j{}^{(l)} + b^{(l)}\right)\right). \tag{12}$$

Furthermore, although the single-head attention mechanism can capture important relationships between nodes by assigning different weights to each node, its limitation lies in potentially insufficient expressive power and vulnerability to noise or outlier data. Therefore, to enhance the model's expressive capability and stability, we introduce a multi-head attention mechanism. The specific formula can be expressed as

$$\boldsymbol{X}_i^{l+1} = \text{concat}\left(\sigma\left(\sum_{j\in\mathcal{N}(i)} \alpha^k{}_{ij}e^k{}_{ij}(\boldsymbol{M}_k{}^{(l)}\boldsymbol{X}_j{}^{(l)} + b_k{}^{(l)})\right)\right),$$

$$k \in (1, 2 \ldots, n), \tag{13}$$

where k represents the number of attention heads. It allows the model to compute multiple independent attention distributions in parallel at the same layer, with each head capturing different relationship patterns or features. When the outputs of these attention heads are combined, the model can not only capture richer information but also improve overall robustness. The specific attention architecture is shown in Fig. 6.

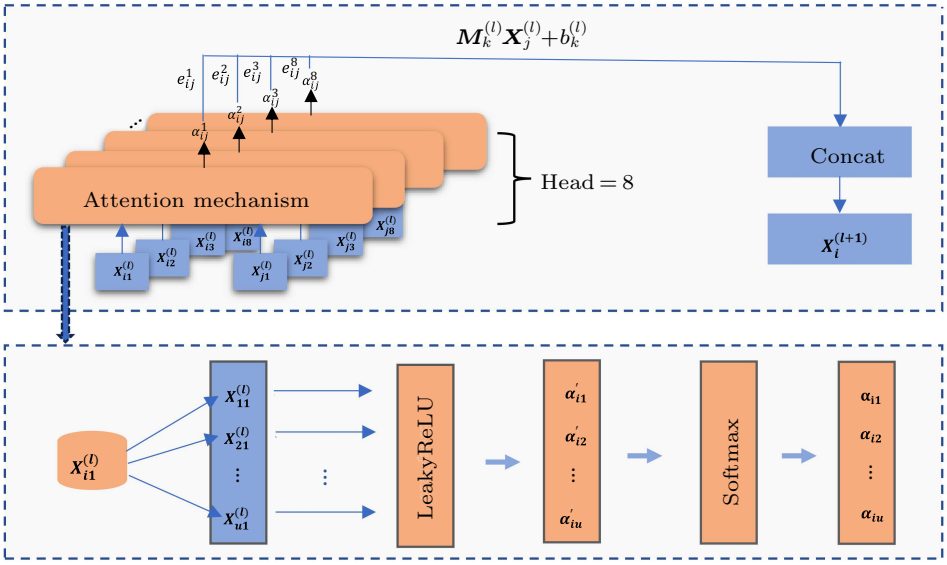


Fig. 5. A schematic diagram of a multi-head graph attention mechanism. The upper part of the diagram shows a typical multi-head attention mechanism with 8 heads ('Head = 8'). Each head calculates different attention weights (such as e_{ij}, α_{ij}), which are then combined through a concatenation (concat) operation to form the updated node representation $\boldsymbol{X}_i^{l+1}$. The lower part of the diagram demonstrates how each attention head calculates attention scores. Initially, the input feature vectors $\boldsymbol{X}_{i1}{}^{l}$ are processed through a LeakyReLU activation function, followed by a normalization using the Softmax function to produce the attention coefficients α_{ij}. These coefficients indicate the attention weights given to the features of each neighboring node.

3.4. Designing decoder process

In GAT model, we first apply a 7-layer multi-head graph attention architecture as described above. The first 6 layers use 8 attention heads to aggregate local neighbor features, while the final layer uses 1 head to generate the final node representations. Through the multi-head attention mechanism, GAT dynamically adjusts the weight relationships between nodes, effectively capturing both local and global error propagation patterns. Each graph attention layer dynamically calculates the influence of neighboring nodes on the current node based on edge weights, ensuring that feature updates flexibly reflect the interactions between errors.

After the graph convolution operation, the output of the GAT is passed through a global mean pooling layer, aggregating all node features into a global graph embedding vector. This embedding vector is then fed into two four-layer fully connected classifiers, which are used to predict logical X and logical Z errors, respectively. Each layer of the classifier incorporates batch normalization to stabilize the training process, and LeakyReLU activation functions are employed to introduce non-linearity. Batch normalization serves to standardize the neuron activations in each mini-batch, ensuring the mean is 0 and the variance is 1, followed by a linear transformation. The mini-batch represents the input from the previous layer. This process is mathematically expressed as

$$y_i = \gamma\left(\frac{x_i - u_{\mathrm{B}}}{\sqrt{\sigma_{\mathrm{B}}^2 + \varepsilon}} + \beta\right), \tag{14}$$

where u_{B} and σ_{B}^2 represent the mean and variance of the mini-batch, respectively, x_i is the feature value of the i-th sample, γ is the scaling parameter that controls the degree of feature amplification, β is the offset parameter that adjusts the feature shift, and ε is a constant used to prevent division by zero. The formulas for calculating the mean u_{B} and variance ${\sigma_{\mathrm{B}}}^2$ are as follows:

$$u_{\mathrm{B}} = \frac{1}{m}\sum_{i=1}^{m} x_i, \tag{15}$$

$$\sigma_{\mathrm{B}}^2 = \frac{1}{m}\sum_{i=1}^{m} (x_i - u_{\mathrm{B}})^2, \tag{16}$$

where m is the number of samples in the mini-batch. Through this standardization process, batch normalization effectively stabilizes the input distribution for each layer, promoting faster model convergence and reducing the risk of vanishing or exploding gradients. The model optimization uses binary cross-entropy loss to measure the error between the predicted results and the target values, and the Adam optimizer is employed to update the model's weights, ensuring efficient convergence and prediction accuracy.

The GAT decoder, through the multi-head attention mechanism and edge weight adjustment, can efficiently handle complex error propagation patterns, making it especially suitable for environments with high physical error rates. The final output of the decoder is the logical error category, determining whether a logical error has occurred in the system. Each time the GAT decoder runs, its output can only be one of the two binary labels (λ_x, λ_y), indicating whether a logical X or logical Z error has occurred. The detailed architecture is shown in Fig. 6.

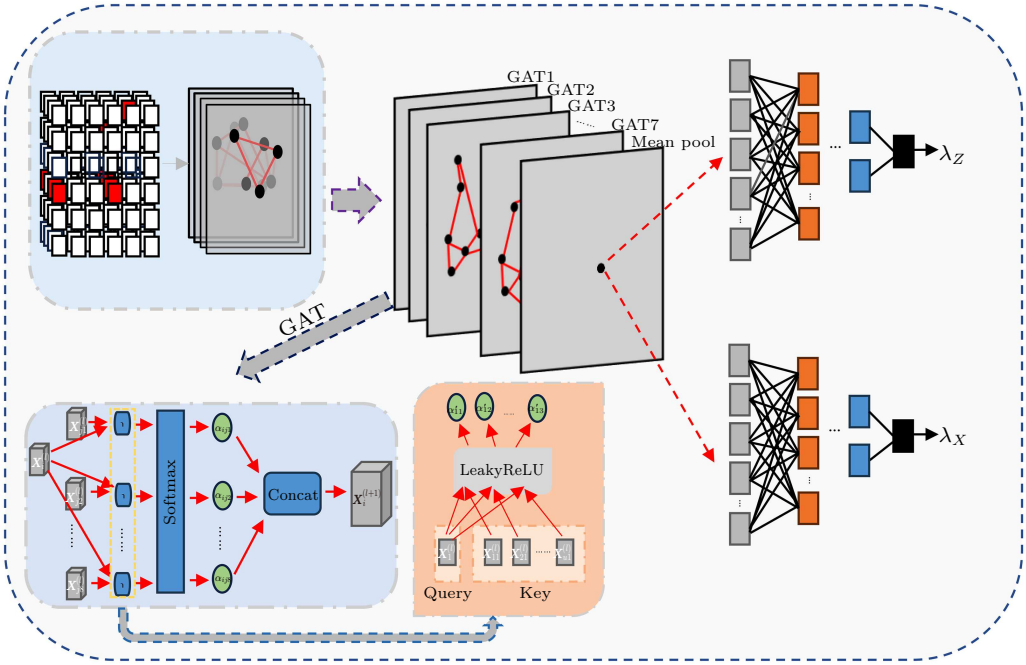


Fig. 6. Structure of the graph attention mechanism and the decoder architecture diagram. The decoder first passes through seven layers of graph attention layers to analyze the data features. Then, the feature data is input into an average pooling layer, which outputs a vector. Finally, depending on the type of errors produced, the data is analyzed by one of two identical four-layer fully connected networks, resulting in an output of 0 or 1, indicating whether a logical error has occurred. The blue section describes the specific steps of the graph attention layer. First, the input node features are linearly transformed to generate query and key vectors, which are used to calculate the attention scores between nodes. These scores are then processed by a LeakyReLU activation function and normalized using the Softmax function to obtain the attention weights. The features of each neighboring node are weighted and aggregated according to these weights, updating the node's feature vector.

3.5. Training process

The training and testing of the GAT decoder are implemented based on the PyTorch geometric library,[46] with an optimization strategy introduced for edge weight computation. The experiments were conducted in a Linux operating system environment using Python version 3.10. To simulate the noise environment in practical quantum computing, we used the stim[47] tool to generate experimental data containing circuit-level noise. These data were generated by introducing Pauli errors with equal probability after Clifford gate operations, before the start of each round, and before and after measurements. In all experiments, the primary focus was on evaluating the decoder's error correction performance when X errors occurred.

The GAT decoder model aims to achieve an error rate below the threshold, thereby improving decoding efficiency and accurately correcting errors. Due to the complexity of circuit-level noise, particularly in high-noise conditions, the model's training faces significant challenges. Therefore, during training, we used error rate data close to the threshold to optimize the model, ensuring it could effectively handle complex noise environments. This strategy helps improve the decoder's accuracy and robustness across different error rates.

To prevent overfitting and improve the model's generalization ability, we replaced 20% of the training data after each training cycle to simulate a larger dataset and ensure model robustness. Specifically, the experimental data were generated using the stim tool, simulating Pauli-X and Pauli-Z stabilizer errors, and represented as 3D syndrome grids. The data included Pauli errors under different noise conditions, and the resulting syndrome data were converted into a graph structure for training the decoder model.

To enhance the model's generalization ability and robustness under various noise conditions, we adopted a dynamic data buffer strategy. After each training cycle, the training data were updated according to a specified buffer size of 5×10^3 and a batch size of 10^3. This training strategy ensured that the model could effectively learn in a circuit-level noise environment, adapt to different noise levels, and avoid the risk of overfitting. This approach not only improved the training efficiency of the GAT decoder but also demonstrated exceptional decoding performance and stability under complex noise conditions.

Furthermore, to efficiently generate training data for larger-scale quantum systems, we propose a comprehensive strategy that integrates Monte Carlo simulation, adaptive data generation, and incremental training to optimize the training of quantum error correction decoders. Firstly, Monte Carlo simulation approximates the target distribution $P(e)$ through random sampling, dynamically generating syndrome data with a focus on representative non-trivial error patterns. The sampling distribution strictly adheres to the error rate $P(e)$ and employs importance sampling to enhance the coverage of samples under high-noise conditions, thereby significantly improving data diversity and relevance.

Secondly, adaptive data generation dynamically produces the most challenging samples based on the model's learning progress, avoiding redundant data and overfitting, and ensuring that the decoder is thoroughly trained on complex error patterns. In this process, the logical error rate is dynamically estimated to assess decoder performance. Meanwhile, incremental training divides the training process into multiple stages, enabling the model to gradually expand the training dataset from an initial small-scale sample set to a fully comprehensive dataset. This phased approach significantly reduces the computational cost of full-data training in large-scale quantum systems. This comprehensive strategy enables real-time estimation of logical error rates and a thorough evaluation of the decoder's robustness and generalization capability using validation sets under different noise conditions. Although this method demonstrates great theoretical potential, further optimization and experimental validation are needed to support efficient training for large-scale quantum systems.

For comparison, the MWPM decoder is implemented using the Pymatching library,[48] with experiments also conducted using Python version 3.10. The experimental data are similarly generated using the stim tool. MWPM is a classical quantum error correction algorithm aimed at predicting the error configuration of qubits by minimizing the total weight of the matching edges, ensuring that this configuration is consistent with the measurements of the detectors. Specifically, the MWPM algorithm maps quantum detectors to the nodes of a graph, with edges between the nodes representing potential matchings. The weight of each edge corresponds to the probability of an error occurring in the corresponding stabilizer. By minimizing the total weight of all matching edges, the algorithm selects the most likely error configuration for correction, thus achieving error correction.

The key of MWPM lies in finding the optimal error configuration through the minimum perfect matching in the graph, that is, by pairing qubits and minimizing the weighted sum of the error probabilities in each pair. The algorithm is fast, especially when the number of qubits and the scale of the experiment increase, as its runtime scales approximately linearly with the size of the graph, making it suitable for large-scale quantum systems. However, as the code distance increases, the computational resource demands of MWPM grow exponentially, which reduces its error correction efficiency. Additionally, MWPM relies on a fixed codeword structure, making it difficult to adapt to different quantum code architectures. Its global search nature significantly increases the computational complexity in large-scale quantum systems, limiting its scalability in practical applications.

In this paper, we compare the MWPM decoder with the GAT decoder. The MWPM decoder relies on classical graph-based matching methods, while the GAT decoder uses graph attention networks for error prediction. Although the MWPM decoder is fast, its adaptability to complex quantum error patterns, especially in high-error-rate and large-scale systems, may not be as strong as that of the GAT decoder.

4. Experimental results

4.1. Training accuracy analysis

After training our GAT decoder, we evaluated its performance. Starting with the trend of training accuracy, figure 7 shows the changes in training and validation accuracy under different code lattice sizes conditions as the number of training epochs increases in the GAT network architecture. Accuracy represents the proportion of correctly predicted samples to the total number of samples, which can be expressed mathematically as

$$\text{Accuracy} = \frac{1}{N}\sum_{i=1}^{N} 1(\hat{y}_i = y_i), \tag{17}$$

where N denotes the number of samples, $\hat{y}_i$ represents the predicted value for the i-th sample, and y_i is the true label for the

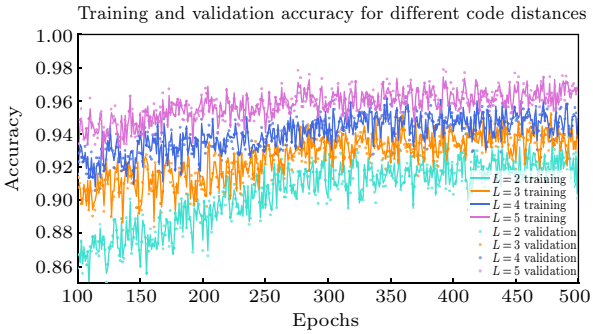


Fig. 7. The figure illustrates the trend of training and validation accuracy for different code lattice sizes ($L=2, L=3, L=4, L=5$) under the GAT network architecture as the number of training epochs progresses. The horizontal axis represents the training epochs, indicating the number of times the model has been trained. After each epoch, the model updates its parameters to enhance its learning capability. The vertical axis denotes the training and validation accuracy, which measures the model's prediction accuracy on the training and validation datasets. Different lines and dots in the figure correspond to the training and validation results for different code distances. The solid lines represent the training accuracy for each code distance, while the dotted lines indicate the validation accuracy for the corresponding code distance.

i-th sample. It can be observed that as the code lattice sizes decreases, the overall accuracy of the model shows a declining trend. However, with the increase in training epochs, the accuracy of models for all code lattice sizes gradually improves and stabilizes, achieving over 0.88 accuracy in most cases.

4.2. Training performance at different repetition counts

To further analyze the performance of the decoder under different conditions, figure 8 presents the training process of the decoder with different repetition counts of $(d_t = 3,5,7,9,11)$. In Fig. 8(a), we observe that as the number of training epochs increases, the decoding accuracy for all repetition counts gradually improves. During the first 100 to 350 training epochs, the accuracy for smaller repetition counts $(d_t = 3,5)$ improves more rapidly, while for larger repetition counts $(d_t = 9,11)$, the improvement is relatively slower. This is because the input graph increases in size not only with code lattice sizes but also with the increase in repetition count, d_t. Fewer repetitions mean that the system is dealing with simpler noise patterns, allowing the model to learn these patterns more quickly, leading to a faster increase in accuracy. However, in the later stages of training (Fig. 8(b), between 350 and 600 epochs), we observe that systems with larger repetition counts start to show more significant performance improvement. As the training progresses, the decoder adapts better to the complex noise and error patterns encountered with larger repetition counts, resulting in a gradual increase in accuracy. Eventually, the accuracy stabilizes and approaches the level of the smaller repetition counts. This indicates that with more training epochs, larger repetition counts provide more noise information, enabling the model to effectively capture and correct more complex error patterns, thereby enhancing overall robustness and accuracy.

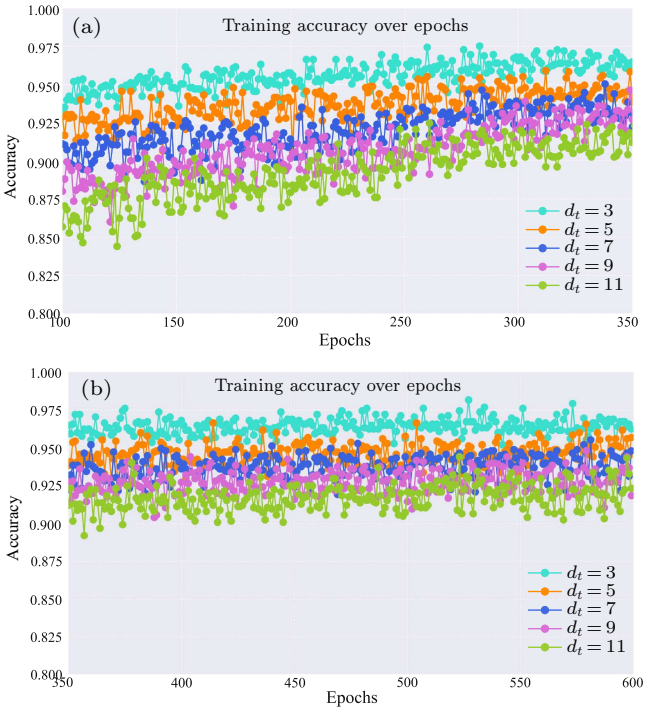


Fig. 8. The trend of training accuracy over training epochs under code lattice sizes $L = 4$ and different repetition counts d_t. Each curve represents the training accuracy for a different repetition count (ranging from 3 to 11). The x-axis represents the number of training epochs, and the y-axis represents accuracy. Subfigure (a) displays the early training phase (100–350 epochs), while subfigure (b) illustrates the later training phase (350–600 epochs). From these figures, it can be observed that the GAT decoder demonstrates high accuracy across various repetition counts, although there are some fluctuations due to the complexity of the noise in the data.

4.3. Comparison and analysis of GAT and MWPM decoders

To further evaluate the performance of the GAT decoder, we compared it with the traditional MWPM decoder, with the results shown in Fig. 9. The figure presents a comparison based on the logical error rate, which indicates the probability that the logical state of the qubit undergoes an error after multiple rounds of stabilizer measurements. The logical error rate is calculated as follows:

$$P_{\text{logical}} = \frac{N_{\text{logical errors}}}{N_{\text{total trials}}}, \tag{18}$$

where P_{logical} represents the logical error rate, $N_{\text{logical errors}}$ denotes the number of logical errors, and $N_{\text{total trials}}$ indicates the total number of trials. A lower logical error rate implies better performance of the error correction code, which is more effective in preserving the integrity of quantum information. In the following, we will provide a detailed analysis of the figure.

At smaller repetition counts $(d_t = 3,5,7)$, the performance of MWPM and GAT is shown in Figs. 9(a) and 9(b). We can see that as the repetition count and code lattice sizes increase, the logical error rate of the MWPM decoder gradually rises. In contrast, the GAT decoder's logical error rate is significantly lower than that of MWPM. This result indicates that the GAT decoder is better at capturing the complex correlations between error chains, leading to more efficient error correction.

The experimental results for larger repetition counts $(d_t = 7,9,11)$ are shown in Figs. 9(c) and 9(d). Compared to smaller repetition counts, the logical error rate of MWPM exhibits a significant upward trend at higher repetition counts, further indicating that as d_t increases, the graph becomes more complex, and the network's convergence becomes increasingly difficult. In contrast, under the same conditions, the logical error rate of the GAT decoder remains at a lower level. This demonstrates that the GAT decoder can maintain its efficient error correction performance when dealing with larger-scale quantum systems, further validating its robustness and generalization capability.

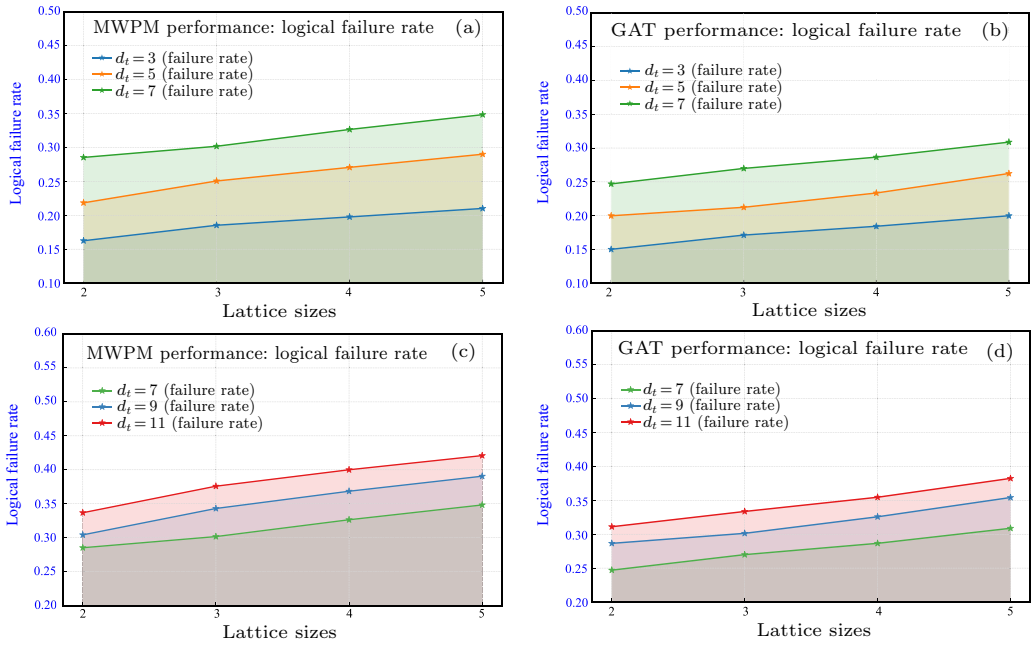


Fig. 9. The figure compares the logical error rates of the GAT decoder and MWPM decoder under different repetition counts d_t. Subfigures (a) and (c) show the performance of the MWPM decoder, while subfigures (b) and (d) present the performance of the GAT decoder. The x-axis represents the code lattice sizes, and the y-axis represents the logical error rate. It can be seen from the figure that the GAT decoder outperforms the MWPM decoder in terms of logical error rate, with the superiority being more prominent at higher repetition counts ($d_t = 9, 11$).

4.4. Threshold analysis

To evaluate the performance of the GAT decoder under different physical error rates, we performed threshold analysis to calculate and determine the decoder's threshold. The threshold is defined as the maximum physical error rate at which the decoder can effectively reduce the logical error rate as the code distance increases. It is an important metric for evaluating the performance of the decoder. Specifically, we conducted multiple experiments or simulations, measuring the logical error rate of the decoder at different physical error rates p and code lattice sizes L, and calculated the corresponding P_{logical} for each physical error rate and code lattice sizes. At low physical error rates, the logical error rate is typically low, but as the physical error rate increases, the logical error rate gradually rises until reaching a critical point. This critical point is the threshold of the decoder.

In Fig. 10, the threshold typically appears as the intersection point of the error curves for different code lattice sizes. To determine this threshold, we simulated quantum systems under various noise conditions and measured the performance of the decoder at different physical error rates. By fitting the error curves and finding their intersection points, we determined the maximum error rate that the decoder can tolerate at different physical error rates p and code lattice sizes L. This threshold is usually represented as a distinct turning point in the graph, indicating that the decoder maintains a low logical error rate below this threshold, but once the physical error rate exceeds this point, the logical error rate begins to rise significantly.

Particularly in large-scale quantum systems, such as the $L = 5$ quantum code, where a large number of qubits and more complex noise models are involved, the simulation and computation complexity increases significantly. To address this challenge, we propose a strategy that combines random sampling, data buffering, and incremental training to optimize the training of quantum error correction decoders. By generating syndrome data and observable flip data through random sampling, and leveraging the characteristics of the error rate distribution $P(e)$, we focus on covering non-trivial error patterns that are more likely to occur. Additionally, we employ a filtering mechanism to remove empty syndromes, retaining only the more informative non-trivial error patterns, thereby significantly enhancing data diversity and effectiveness. Under the assumption of an independent error model, the error rate distribution $P(e)$ can be expressed mathematically as

$$P(e) = \prod_{i=1}^{n} p_i^{e_i} (1 - p_i)^{1 - e_i},$$

where n is the total number of qubits, $e_i = 1$ indicates that the i-th qubit has an error, $e_i = 0$ indicates no error, and p_i is the error rate of the i-th qubit. This formula describes the probability of each error pattern under the assumption of independent errors.

To avoid overfitting and adapt to different noise conditions, we dynamically update a portion of the data buffer in each training iteration, replacing older data with newly generated syndrome data. This dynamic updating mechanism allows the decoder to gradually adapt to the error pattern distributions under various noise conditions. Furthermore, to ensure the generalization ability and robustness of the model, we periodically evaluate its performance on an independent validation set. The validation set covers different noise levels,

and the decoder's performance is assessed using logical error rates. This strategy, which integrates random sampling, dynamic data updating, and incremental training, not only optimizes the threshold calculation of the decoder but also significantly improves its decoding performance and stability under complex noise conditions, providing strong support for efficient training in large-scale quantum systems.

Through these experiments and analyses, we compared the thresholds of the GAT decoder and the MWPM decoder under different physical error rates. The results show that the threshold of the GAT decoder is significantly higher than that of the MWPM decoder. Specifically, the threshold of the MWPM decoder is approximately 0.69%, while the threshold of the GAT decoder is approximately 0.78%. When the physical error rate is below the threshold, the logical error rate P_{logical} of the GAT decoder decreases significantly as the code lattice sizes increases, and its behavior can be approximately described by

$$P_{\text{logical}}(p, L) \approx C \cdot p^k,$$

where C is a constant related to the error correction code, and k is a parameter that increases with the code lattice sizes L. This indicates that at low physical error rates, increasing the code lattice sizes can significantly reduce the logical error rate, demonstrating that the GAT decoder has stronger error-correction capability in low physical error rate environments.

When the physical error rate exceeds the threshold, the logical error rate increases with the physical error rate. However, the rate of increase for the GAT decoder is significantly smaller than that of the MWPM decoder. This indicates that even in high physical error rate environments, the GAT decoder is able to maintain a lower logical error rate, demonstrating stronger noise resilience, and further proving its significant advantage in complex noise environments.

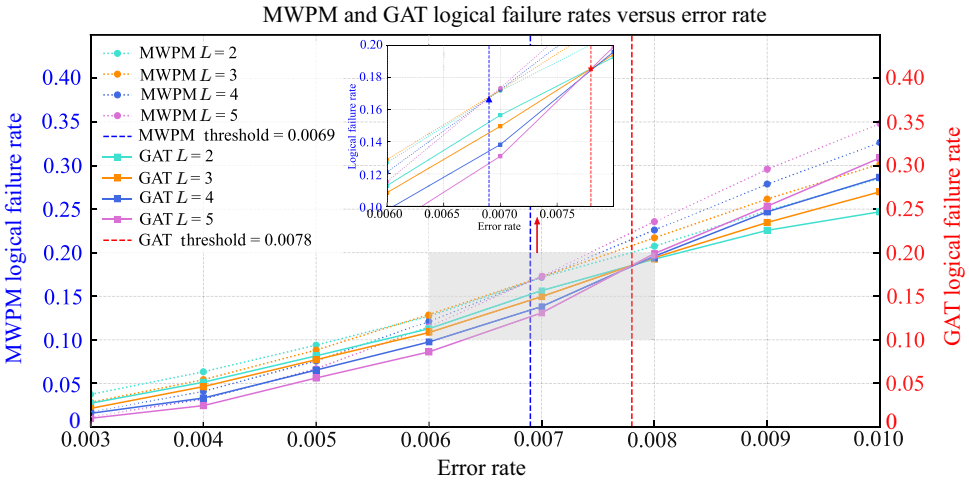


Fig. 10. The relationship between the logical error rate (left axis in blue for MWPM, right axis in red for GAT) and the physical error rate for different lattice sizes $L = 2, 3, 4, 5$. As the physical error rate increases, the logical error rate for both methods gradually rises. The dashed lines indicate the thresholds for the MWPM and GAT decoders, where the threshold for MWPM is 0.0069 and for GAT is 0.0078, indicating that GAT can maintain a lower logical error rate at higher physical error rates compared to MWPM, with a higher threshold.

In conclusion, the experimental results demonstrate that the GAT decoder outperforms the traditional MWPM decoder in terms of logical error rate, decoding accuracy, and noise resistance. Through the use of a multi-head attention mechanism and edge weight adjustment, the GAT model can flexibly handle complex quantum error chains, showing significant performance improvement, particularly in large-scale quantum systems. Additionally, the GAT decoder has relatively lower computational complexity and is well-suited for large-scale parallel computation, making it a potential solution for quantum error correction in future large-scale quantum computing systems.

5. Conclusion

We proposed a GAT decoder for decoding SSC with three-qubit check operators. Experimental results demonstrate that the GAT decoder performs exceptionally well in handling circuit-level noise in quantum computing. Compared to traditional MWPM decoders, the GAT decoder not only improves decoding efffciency but also, in some cases, increases the logical error threshold, showing higher decoding accuracy and reliability. By incorporating attention mechanisms and a multi-layer graph attention structure, the GAT decoder effectively captures and utilizes the complex relationships in error syndromes, achieving efffcient decoding across different code lattice sizes and error rates.

In future work, the GAT decoder has broad prospects for practical quantum computing applications. Its parallel computation capabilities and ffexibility make it well-suited for handling large-scale quantum error correction tasks. Additionally, the architecture of the GAT decoder is highly scalable and can be adapted to other types of quantum error correction codes, such as color codes and hexagonal stabilizer codes, laying a

foundation for its application in a wider range of quantum computing tasks. Future research will continue to optimize the performance of the GAT decoder and explore its application in different quantum computing platforms and larger-scale subsystem codes to advance the development of quantum computing technology.

Acknowledgements

Project supported by the Natural Science Foundation of Shandong Province, China (Grant No. ZR2021MF049), the Joint Fund of the Natural Science Foundation of Shandong Province, China (Grant Nos. ZR2022LLZ012 and ZR2021LLZ001), and the Key Research and Development Program of Shandong Province, China (Grant No. 2023CXGC010901).

References

[1] Song Z Y, Wang Y N, Wang H W, *et al.* 2023 *Acta Electronica Sinica* **51** 2030
[2] Gong L H, *et al.* 2024 *Laser Phys. Lett.* **21** 055209
[3] Zhou N R, *et al.* 2024 *Advanced Quantum Technologies* **7** 2400530
[4] Gong L H, *et al.* 2023 *Advanced Quantum Technologies* **6** 2300097
[5] Chen Q H, *et al.* 2024 *Chin. Phys. B* **33** 060314
[6] Wang H W,Xue Y J, Qu Y J, *et al.* 2022 *npj Quantum Inf.* **8** 134
[7] Bravyi S, and Kitaev A Y 1998 arXiv preprint **52** 9811
[8] Bravyi S, Duclos-Cianci G, Poulin D, *et al.* 2012 arXiv preprint **1207** 1443
[9] Monz T, Kim K, Villar A S, *et al.* 2008 *Nature* **453** 1008
[10] Cui T H, Wei Y Q, Li J, *et al.* 2024 *Chin. Phys. B* **33** 043701
[11] Cai X, Zhou Y B, Yu W L, *et al.* 2024 *Chin. Phys. B* **33** 068501
[12] Dennis E, Kitaev A, Landahl A, *et al.* 2021 *J. Math. Phys.* **43** 4452
[13] Fowler A G, Whiteside A C and Hollenberg L C L 2012 *Phys. Rev. A* **86** 042317
[14] Heim B, Svore K M and Hastings M B 2016 arXiv preprint **1609** 06373
[15] Fowler A G, Mariantoni M, Martinis J M, *et al.* 2012 *Phys. Rev. A* **86** 032324
[16] Veličković P, Cucurull G, Casanova A, *et al.* 2017 arXiv preprint **1710** 10903
[17] Yang P, Lu P and Zhang T 2023 *Chin. Phys. B* **32** 058902
[18] Vaswani A, *et al.* 2017 arXiv preprint **1706** 03762
[19] Lange M, Havström P, Srivastava B, *et al.* 2023 arXiv preprint **2307** 01241
[20] Lange M 2023 *Decoding the Surface Code Using Graph Neural Networks* Master's Thesis (Gothenburg: University of Gothenburg, Department of Physics)
[21] Bergentall V 2021 *Quantum Error Correction Using Graph Neural Networks* Master's Thesis (Gothenburg: University of Gothenburg, Department of Physics)
[22] Gao Z H, Zhu C S and Wang C L 2023 *Chin. Phys. B* **32** 078101
[23] Li A Q, Li F, Gan Q D, *et al.* 2023 *Appl. Sci.* **13** 9689
[24] Fan X K, Yan D D, Liu F, *et al.* 2022 *Sci. Sin. Inform.* **25** 539
[25] Li D , Wang H W, Wang Y N, *et al.* 2022 *Quantum Engineering* **2022** 9638108
[26] Li F, Li A Q, *et al.* 2024 *Chin. Phys. B* **33** 040307
[27] Wang H W, Song Z Y, Wang Y N, *et al.* 2022 *Quantum Inf. Process.* **21** 280
[28] Qu Y J, Chen Z, *et al.*.2023 *Chin. Phys. B* **32** 100307
[29] Wang H W, Xue Y J, Ma Y L, *et al.* 2022 *Chin. Phys. B* **31** 010303
[30] Wilde M M 2009 *Phys. Rev. A* **79** 062322
[31] Ji Y X, Chen Q H, Wang R, *et al.* 2024 *Quantum Information Processing* **23** 255
[32] Ma H Y, Xu P A, Shao C H, *et al.* 2019 *Int. J. Theor. Phys.* **58** 4241
[33] Huber F and Grassl M 2020 *Quantum* **4** 284
[34] Liu B, Wang Y X 2024 *Chin. Phys. B* **33** 084401
[35] Kipf T N and Welling M 2016 arXiv preprint **1609** 02907
[36] Zhang Y, Wang X, Wang Y, *et al.* 2024 *Chin. Phys. B* **33** 070702
[37] Zhou J, Cui G, Hu S, *et al.* 2020 *AI Open* **1** 57
[38] Cheng T, Zhao R S, Wang S, *et al.* 2024 *Chin. Phys. B* **33** 040303
[39] Simonyan K and Zisserman A 2014 arXiv preprint **1409** 1556
[40] Xu S, Zhu X, Wang J, *et al.* 2021 *Chin. Phys. B* **30** 048402
[41] Shao X G, Zhang J and Lu Y J 2024 *Chin. Phys. B* **33** 070203
[42] Hewamalage H, Bergmeir C and Bandara K 2021 *International Journal of Forecasting* **37** 388
[43] Huang Y J and Hu H G 2015 *Chin. Phys. B* **24** 120701
[44] Grattarola D, Zambon D, Bianchi F M, *et al.* 2022 *IEEE Trans. Neural Networks and Learning Systems* **35** 2708
[45] Liu C, Zhan Y B, *et al.* 2022 arXiv preprint **2204** 07321
[46] Fey M and Lenssen J E 2019 arXiv preprint **1903** 02428
[47] Gidney C 2021 *Quantum* **5** 497
[48] Higgott O 2022 *ACM Transactions on Quantum Computing* **3** 1