# DyREM: Dynamically Mitigating Quantum Readout Error with Embedded Accelerator

Kaiwen Zhou
*Zhejiang University*
Hangzhou, China
kaiwenzhou@zju.edu.cn

Liqiang Lu*
*Zhejiang University*
Hangzhou, China
liqianglu@zju.edu.cn

Hanyu Zhang
*Zhejiang University*
Hangzhou, China
hyzz@zju.edu.cn

Debin Xiang
*Zhejiang University*
Hangzhou, China
db.xiang@zju.edu.cn

Chenning Tao
*Zhejiang University*
Hangzhou, China
tcn@zju.edu.cn

Xinkui Zhao
*Zhejiang University*
Hangzhou, China
zhaoxinkui@zju.edu.cn

Size Zheng
*ByteDance Ltd*
Beijing, China
zheng.size@bytedance.com

Jianwei Yin
*Zhejiang University*
Hangzhou, China
zjuyjw@zju.edu.cn

2025 62nd ACM/IEEE Design Automation Conference (DAC) | 979-8-3315-0304-8/25/$31.00 ©2025 IEEE | DOI: 10.1109/DAC63849.2025.11132635

***Abstract*—Quantum readout error is the most significant source of error, substantially reducing the measurement fidelity. Tensor-product-based readout error mitigation has been proposed to address this issue by approximating the mitigation matrix. However, this method inevitably encounters the dynamic generation of the mitigation matrix, leading to long latency. In this paper, we propose DyREM, a software-hardware co-design approach that mitigates readout errors with an embedded accelerator. The main innovation lies in leveraging the inherent sparsity in the nonzero probability distribution of quantum states and calculating the tensor product on an embedded accelerator. Specifically, using the output sparsity, our dataflow dynamically downsamples the original mitigation matrix, which dramatically reduces the memory requirement. Then, we design DyREM architecture that can flexibly gate the redundant computation of nonzero quantum states. Experiments demonstrate that DyREM achieves an average speedup of $9.6\times \sim 2000\times$ and fidelity improvements of $1.03\times \sim 1.15\times$ compared to state-of-the-art readout error mitigation methods.**

## I. Introduction

In the current Noisy Intermediate-Scale Quantum (NISQ) era, various sources of errors hinder the full realization of quantum computing advantage [1], [2]. Among these, readout error is a significant source of noise, with error rates ranging from 1% to 10% on state-of-the-art superconducting quantum processors [2]–[4]. Consequently, mitigating readout error is essential to ensure high fidelity in both NISQ quantum processors and future fault-tolerant quantum systems.

Several mitigation approaches have been proposed, including matrix-based mitigation [5]–[7], machine learning [8], and measurement subsetting [9]. Among these, matrix-based mitigation is widely used, employing matrix-vector multiplication (MVM) to improve measurement fidelity. However, this approach is computationally intensive as the mitigation matrix size grows exponentially with the number of qubits [10]. To address the memory issue, the tensor-product-based approach provides a more general formulation by decomposing the mitigation matrix into a tensor product of sub-mitigation matrices. Each sub-mitigation matrix is independently characterized based on the specific subset of qubits. This formulation avoids constructing and storing the full-dimensional mitigation matrix, making it inherently scalable.

*Corresponding author

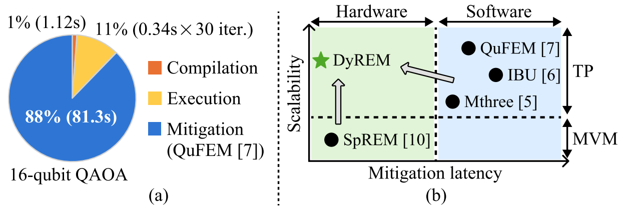


Fig. 1. (a) Breakdown of end-to-end latency for the 16-qubit QAOA on the 156-qubit $ibm\_fez$ quantum processor. (b) The tradeoff between scalability and latency for MVM-based method SpREM [10] and tensor-product-based (TP) methods Mthree [5], IBU [6], and QuFEM [7].

However, current tensor-product-based approaches still face limitations, including low accuracy, insufficient flexibility, and high latency. For example, Mthree [5] and IBU [6] calculate the mitigation matrix using $2 \times 2$ qubit-independent sub-mitigation matrices. However, these $2\times2$ sub-mitigation matrices fail to consider qubit crosstalk, severely reducing fidelity. To improve fidelity, QuFEM [7] groups physical qubits and utilizes grouping matrices to approximate mitigation matrix. However, QuFEM lacks the adaptability to varying measured qubits, as its grouping matrices are static and require pre-determination. Furthermore, the inherently time-intensive nature of tensor product operations results in significant latency across these approaches. For example, as shown in Figure 1(a), the mitigation time of QuFEM accounts for 88% of the end-to-end latency for a 16-qubit QAOA circuit. Overall, as shown in Figure 1(b), current methods face a tradeoff between scalability and mitigation latency.

In this paper, we propose DyREM, a software-hardware co-design approach that dynamically mitigates readout errors with an embedded accelerator. The key innovation of DyREM lies in exploiting the inherent sparsity in the nonzero probability distribution of quantum states and calculating the tensor product on an embedded accelerator. To adapt to varying measured qubits, our dataflow dynamically downsamples the original mitigation matrix to obtain the required mitigation matrix. Then, we introduce nonzero state-oriented computation to compress and calculate the mitigation matrix efficiently. Finally, we design DyREM architecture to support our dataflow, which can flexibly gate the redundant computation of nonzero quantum states.

The contribution of this paper can be summarized as follows:

- We propose DyREM, a software-hardware co-design approach that dynamically mitigates readout error with an embedded accelerator, leveraging the inherent sparsity in the nonzero probability distribution of quantum states.
- We introduce a dataflow that dynamically downsamples the original mitigation matrix, dramatically reducing the memory requirement. We further design DyREM architecture that flexibly gates the redundant computation of nonzero quantum states.
- We evaluate the effectiveness of DyREM by comparing it with state-of-the-art readout error mitigation methods. The results demonstrate the superiority of DyREM, achieving an average of $9.6\times \sim 2000\times$ speedup and $1.03\times \sim 1.15\times$ fidelity improvement.

## II. Background

### A. Matrix-Based Readout Error Mitigation

After the measurement operation, the readout error maps the ideal probability distribution $P_{ideal}$ into a noise probability distribution $P_{noisy}$. This process can be mathematically represented as a linear transformation, where $P_{ideal}$ is multiplied by a noise matrix. To mitigate readout error, we need to multiply $P_{noisy}$ by a mitigation matrix $M$:

$$P_{miti} = M \cdot P_{noisy} \tag{1}$$

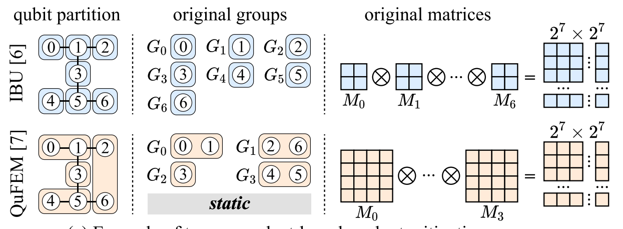
where the size of $M$ is $2^n \times 2^n$ ($n$ refers to the number of qubits). After the mitigation, the probabilities of the ideal states are expected to increase, while those of the noisy probabilities are reduced. However, the size of $M$ scales exponentially with $n$, exhibiting poor scalability. For example, a 30-qubit mitigation matrix requires 32,768 PB of memory, which is $46.8\times$ the capacity of the world's fastest supercomputer Frontier (700 PB) [11].

### B. Tensor-Product-Based Readout Error Mitigation

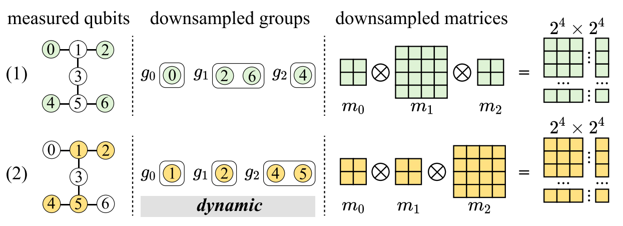
To address the memory issue, tensor-product-based readout error mitigation has been proposed. The key idea is to approximate the full-dimensional mitigation matrix $M$ using a series of tensor-product operations [5]–[7]:

$$P_{miti} = (M_1 \otimes M_2 \otimes \cdots \otimes M_k) \cdot P_{noisy} \tag{2}$$

where $M_1, M_2, \cdots, M_k$ are sub-mitigation matrices obtained by partitioning the measured qubits into $k$ groups. As illustrated in Figure 2(a), IBU [6] regards each physical qubit as an independent unit and approximates $M$ using a tensor product of seven $2 \times 2$ mitigation matrices (i.e., $M_0 \sim M_6$). However, IBU achieves only limited mitigation accuracy because it neglects the measurement crosstalk between qubits.



(a) Example of tensor-product-based readout mitigation process.



(b) Generalized mitigation matrix generation according to the measured qubits.

Fig. 2. (a) Tensor-product-based readout mitigation process. (b) Generalized mitigation matrix generation by dynamically computing the downsampled groups.

To consider crosstalk effects, QuFEM [7] proposes a more general formulation by grouping physical qubits to approximate $M$ accurately. As shown in Figure 2(a), seven qubits on the device are partitioned into four groups (i.e., $G_0 \sim G_3$) by quantifying the interactions between qubits. Each group is then characterized to produce its own mitigation matrix (i.e., $M_0 \sim M_3$). However, the groups and their mitigation matrices are ***static*** as they must be pre-determined before the mitigation process begins.

### C. Motivation

In practical quantum circuit execution, the set of measured qubits varies due to differing levels of transpiler optimization, as shown in Figure 2(b). Consequently, the groups of measured qubits are dynamic, rendering the pre-determined mitigation matrices unsuitable for reuse during mitigation.

This limitation motivates us to develop a new approach to enable dynamic readout error mitigation. To this end, we introduce the concept of *downsampled group*, denoted by $g_i$. We dynamically determine the downsampled group $g_i$ based on the measured qubits and original groups $G_j$. Then, we calculate the downsampled matrices $m_i$ for $g_i$ by deriving them from the original matrices $M_j$. Finally, we calculate the mitigation matrix based on the downsampled matrices $m_i$ and perform the MVM step to complete mitigation.

## III. DyREM Dataflow

### A. Mitigation Dataflow Overview

Figure 3 illustrates an example of DyREM mitigation dataflow. The input consists of measured qubits and noisy distribution. The output is mitigated distribution $P_{miti}$. The dataflow is divided into three steps. **Step 1.** downsampled groups generation: we partition the measured qubits '①②④⑤' according to the original groups $G_0 \sim G_3$. Then, we obtain four downsampled groups $g_0 \sim g_3$, which serve as the input for matrix downsampling. **Step 2.** matrix downsampling: we categorize the downsampled groups

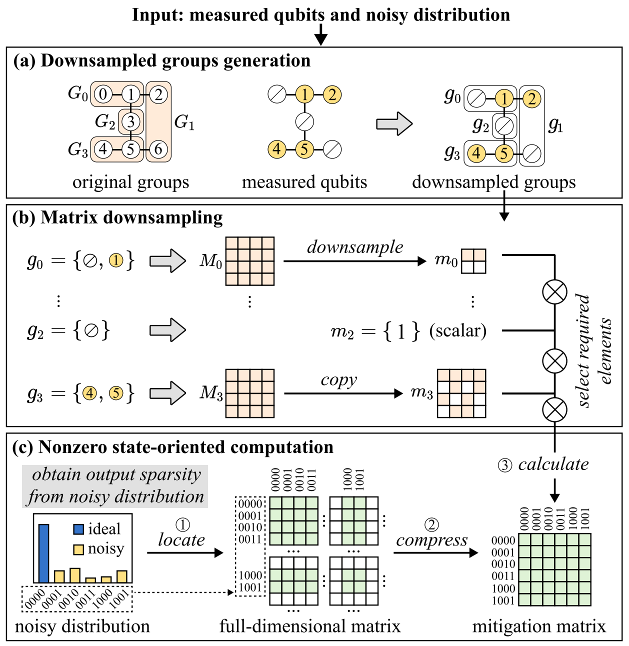


Fig. 3. Example of DyREM mitigation dataflow.

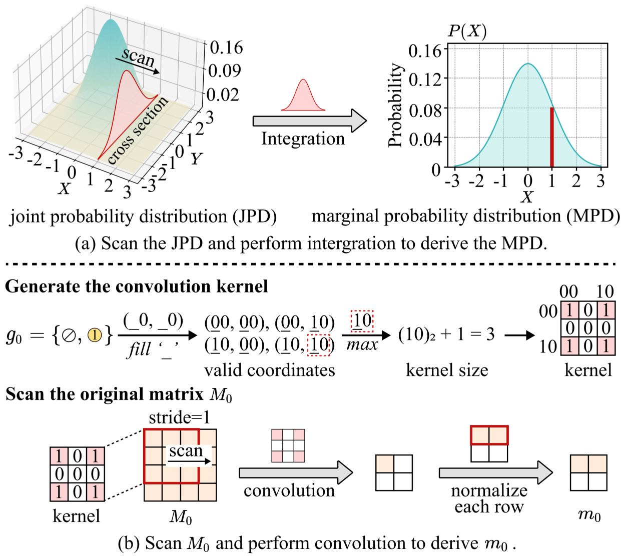

scan
cross section
Integration
joint probability distribution (JPD)
marginal probability distribution (MPD)
(a) Scan the JPD and perform intergration to derive the MPD.
Generate the convolution kernel
valid coordinates
kernel size
kernel
Scan the original matrix M₀
stride=1
convolution
normalize each row
(b) Scan M₀ and perform convolution to derive m₀ .

Fig. 4. Inspiration and an example of the matrix downsampling for partially measured groups.

$g_0 \sim g_3$ into three types. Then, we adopt three strategies to generate the downsampled matrices $m_0 \sim m_3$, serving as the input for nonzero state-oriented computation. **Step 3.** nonzero state-oriented computation: we first obtain the output sparsity from the noisy distribution. Then, we compress the full-dimensional matrix based on the output sparsity. Next, we calculate the mitigation matrix by selecting the required elements from the downsampled matrices $m_0 \sim m_3$ and performing multiplication. Finally, we perform matrix-vector multiplication to mitigate the readout error.

In the following sections, we explain the details of **step 2** and **step 3**, respectively.

## *B. Matrix Downsampling*

As depicted in Figure 3(a), some physical qubits are not measured (denoted by $\oslash$) in circuit execution. Therefore, the downsampled groups $g_0 \sim g_3$ are not identical to the original groups $G_0 \sim G_3$. To consider this variation, we categorize the downsampled groups into three types: (1) unmeasured (e.g., $g_2$), (2) fully measured (e.g., $g_3$), and (3) partially measured (e.g., $g_0$ and $g_1$). As shown in Figure3(b), we employ tailored strategies to generate downsampled matrices for each type of group. For 'unmeasured' $g_2$, we set its downsampled matrix to scalar '1', which does not influence the computational result. For 'fully measured' $g_3$, we directly copy the original matrix $M_3$ to obtain the downsampled matrix $m_3$ without additional computation. For 'partially measured' $g_0$ and $g_1$, we apply a convolution kernel to downsample the original matrix.

Figure 4 presents the downsampling of the original matrix $M_0$. Figure 4(a) illustrates the process of deriving the marginal probability distribution (MPD) of variable $X$ from the joint probability distribution (JPD) of two variables $X, Y$. By taking cross sections of the JPD along the $x$-axis and performing integration, the probability distribution of variable $X$ is obtained. This process inspires us to use convolution kernels to downsample the original matrix, as the original matrix $M_0$ essentially represents the JPD of discrete variables (e.g., qubits). Figure 4(b) presents the detailed process. To obtain the convolution kernel, we first generate the valid coordinates based on $g_0$, where the value at measured qubit ① is fixed at 0, and the value at $\oslash$ is filled sequentially from '00' to '11'. The values at the valid coordinates are set to 1, while all other positions are set to 0. Then, we extract the maximal sub-coordinate '10', convert it to decimal, and add one to obtain the kernel size. Next, we determine the stride based on $g_0$ by setting $\oslash$ to 0 and measured qubit ① to 1. We then convert '01' to decimal to obtain the stride (e.g., 1). Finally, we scan $M_0$ using the kernel and normalize each row to obtain the downsampled matrix $m_0$.

## *C. Nonzero State-Oriented Computation*

To enable on-chip calculation of the mitigation matrix, we propose nonzero state-oriented computation, comprising two steps: mitigation matrix compression and nonzero state similarity detection.

**Mitigation matrix compression.** A straightforward approach to obtain the mitigation matrix is to compute the tensor product of the downsampled matrices $m_0$ to $m_3$. However, this approach is time-consuming and resource-intensive. We observe that the ideal distribution usually lies in the noisy distribution, which only contains a few nonzero values. Therefore, we exploit this output sparsity from the noisy distribution to compress the mitigation matrix. By confining the mitigation process to the noisy space, we reduce the mitigation matrix size from exponential to linear. For example, in Figure 3(c), the mitigation matrix size is reduced from $16 \times 16$ to $6 \times 6$ based on the output sparsity.

**Nonzero state similarity detection.** After compression, we cannot simply perform a tensor product of downsampled matrices to calculate the mitigation matrix. Figure 5 shows

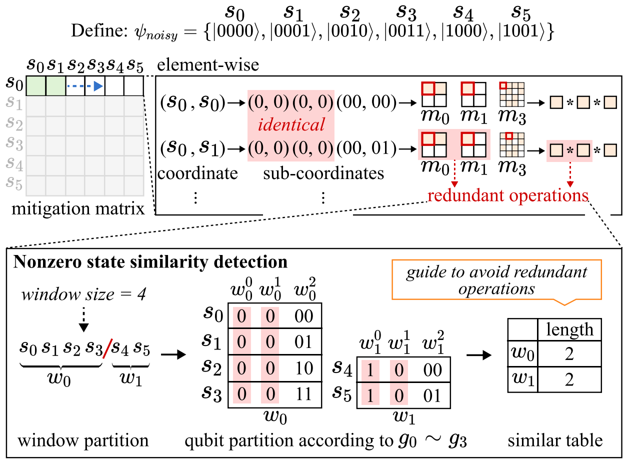


Fig. 5. The element-wise calculation process of mitigation matrix and the illustration of nonzero state similarity detection.

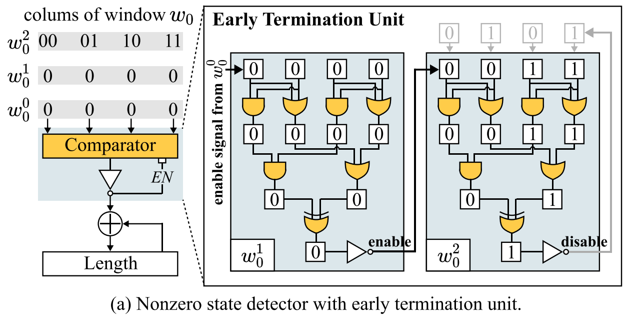
a straightforward approach to compute the mitigation matrix elements individually: for each element, its coordinates are divided into sub-coordinates based on the downsampled groups $g_0 \sim g_3$; the corresponding elements are then extracted from the downsampled matrices $m_0 \sim m_3$ and multiplied cumulatively to obtain the matrix element. However, we find that this approach involves many redundant operations. For example, the first two sub-coordinates of $(s_0, s_0)$ and $(s_0, s_1)$ are identical, which means the extraction and multiplication for $(s_0, s_1)$ are redundant.

To eliminate redundant operations, we introduce nonzero state similarity detection. Specifically, we use a window to partition the noisy states sequentially. Note that we assume the noisy states are ordered according to their binary values. Then, we partition the qubits of noisy states according to the downsampled groups $g_0 \sim g_3$. Within each window, we identify the longest identical segment and obtain its length to construct a similar table. We can generate this table once we receive the noisy distribution. Then, we utilize this table to guide the calculation of the mitigation matrix, thereby eliminating redundant operations. Accordingly, we compute each row of the mitigation matrix window-wise rather than element-wise.

Figure 5 shows the process of computing a similar table for a mitigation matrix. First, we set the window size to 4. Then, the noisy states are partitioned into two windows (i.e., $w_0$ and $w_1$). In each window, the noisy states are partitioned into three columns according to the downsampled groups $g_0 \sim g_3$. Clearly, in $w_0$, the largest identical segment is $(w_0^0, w_0^1)$, while in $w_1$, it is $(w_1^0, w_1^1)$. Next, we record the length of these two segments in a similar table (i.e., 2 and 2). After applying the nonzero state similarity detection, we reduce the extraction and multiplication operations required for $6 \times 6$ mitigation matrix calculation by 44% and 33%, respectively.

## IV. Architecture Design

We design an accelerator architecture to implement DyREM, primarily consisting of two components: a nonzero state detector and a mitigation core array. The nonzero state detector efficiently calculates the length of the longest identical segment for each window. The mitigation core array receives the values from downsampled matrices and noisy probabilities. Each mitigation core (MC) employs a three-layer multiplier to integrate the mitigation matrix calculation with the MVM step.


(a) Nonzero state detector with early termination unit.

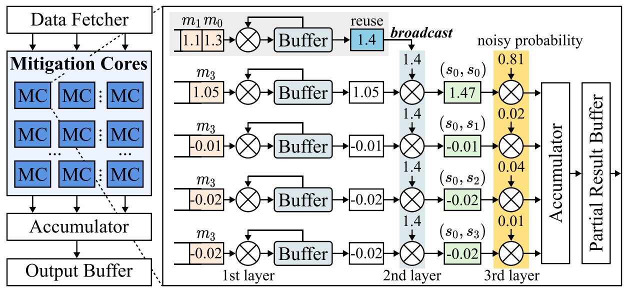

(b) The mitigation module with dedicated mitigation core.

Fig. 6. The hardware design details of DyREM.

### A. Nonzero State Detector

As shown in Figure 6(a), the nonzero state detector integrates an early termination unit with an adder. The early termination unit consists of a comparator and a NOT gate. The comparator receives the columns of each window and determines whether the bits in each column are identical. The comparator consists of a series of logic gates (AND, NOT, and XOR gates), forming a hierarchical tree. If the comparator outputs '0', it indicates that all bits in the column are identical; otherwise, they are not. Specifically, the output then passes through a NOT gate to generate an enable signal, indicating whether the subsequent comparison should proceed. In other words, if the bits in the previous column are not identical, there is no need to perform further comparisons. After each comparison, the adder receives the output from the NOT gate, accumulating them to obtain the identical table.

Figure 6(a) shows an example of the nonzero state detector operating on window $w_1$. The comparator first receives the bits in $w_0^0$ and determines they are identical, enabling the comparison of $w_0^1$. Meanwhile, the length increases by one. Similarly, the comparison result of $w_0^1$ enables the comparison of $w_0^2$, while incrementing the length to 2. Since each element in $w_0^2$ contains two bits, we divide $w_0^2$ into two parts and sequentially input them into the comparator. As shown in Figure 6(a), the first part is not identical. Thus, the output of the comparator is 1, disabling the comparison of the second part. Consequently, the length is incremented by 0. Finally, the length is 2, indicating $w_1$ has two identical columns.

TABLE I
THE HARDWARE PERFORMANCE COMPARISON.

| Baseline | Technical feature | VQE [12] |  | QAOA [13] |  | DJ [14] |  |
|---|---|---|---|---|---|---|---|
|  |  | Latency (s) | Q-throughput (states/s) | Latency (s) | Q-throughput (states/s) | Latency (s) | Q-throughput (states/s) |
| **Mthree** [5] | Hamming pruning | 2.52 (384×) | $7.27 \times 10^3$ (583×) | 4.22 (461×) | $4.63 \times 10^3$ (758×) | 0.66 (206×) | $2.77 \times 10^4$ (192×) |
| **SpREM** [10] | HDSR format | 0.48 (73.8×) | $1.76 \times 10^6$ (2.4×) | 0.56 (61.5×) | $1.49 \times 10^6$ (2.3×) | 0.031 (9.6×) | $3.43 \times 10^6$ (1.5×) |
| **IBU** [6] | Bayesian unfolding | 13.0 (2000×) | $3.58 \times 10^5$ (11.8×) | 17.1 (1879×) | $2.72 \times 10^5$ (12.9×) | 4.59 (1437×) | $1.01 \times 10^6$ (5.2×) |
| **QuFEM** [7] | Finite element analysis | 11.7 (1800×) | $1.56 \times 10^3$ (2726×) | 13.5 (1483×) | $1.45 \times 10^3$ (2420×) | 5.42 (1687×) | $2.65 \times 10^3$ (2002×) |
| **DyREM** | Redundancy detection | $\mathbf{6.52 \times 10^{-3}}$ | $\mathbf{4.24 \times 10^6}$ | $\mathbf{9.12 \times 10^{-3}}$ | $\mathbf{3.51 \times 10^6}$ | $\mathbf{3.25 \times 10^{-3}}$ | $\mathbf{5.31 \times 10^6}$ |

### *B. Mitigation Core*

As depicted in Figure 6(b), the mitigation module comprises a data fetcher, a mitigation core array, an accumulator, and an output buffer. The data fetcher receives the similar table and the noisy distribution. Then, it extracts the values from the downsampled matrices and passes them to the mitigation core array. We allocate 4KB of on-chip SRAM for the downsampled matrices to support quick random accesses, with each element stored in FP16 format.

Each MC is responsible for a single window, including calculating the mitigation matrix elements and performing the mitigation. Within the MC, we set up a FIFO for each multiplier (1st layer) to stockpile the values from downsampled matrices. In particular, the FIFO in the first row is used to calculate the reused data. Each remaining FIFO corresponds to the calculation of a mitigation matrix element. After obtaining the reused data, we broadcast it to the multipliers in the 2nd layer. Then, we perform multiplication to calculate the mitigation matrix elements within the window. Finally, we execute mitigation by multiplying noisy probabilities and the mitigation matrix elements. The accumulator receives the results from each multiplier and sends the accumulated result to the partial result buffer. The results of the MCs within a row are accumulated to yield one mitigated probability.

Figure 6(b) illustrates an example of the MC processing $w_0$. The values 1.1 and 1.3 correspond to the sub-coordinates $(0, 0)$ and $(0, 0)$ of $(s_0, s_0)$ in Figure 5, respectively. The multiplication result 1.4 is reused to compute mitigation matrix elements within $w_0$. After obtaining the values of $(s_0, s_0) \sim (s_0, s_3)$, we perform multiplication with noisy probabilities and accumulate the results to obtain the mitigated result of $w_0$.

## V. EVALUATION

### *A. Evaluation Setup*

**Hardware implementation.** We develop DyREM accelerator using Xilinx High-Level Synthesis (HLS) C++ and implement it with Vitis 2023.1. We evaluate its hardware performance on the Xilinx U50 FPGA, operating at 300 MHz.

**Benchmarks.** The benchmarks include Variational Quantum Eigensolver (VQE) [12], Quantum Approximate Optimization Algorithm (QAOA) [13], Feedback-Based Quantum Optimization (FALQON) [15], and Deutsch-Jozsa (DJ) [14] algorithms, with two layers set for both VQE and QAOA.

**Baselines.** We compare DyREM with state-of-the-art readout error mitigation methods, including Mthree [5], SpREM [10], IBU [6] and QuFEM [7]. For Mthree, SpREM, and IBU, we set the Hamming distance to 3. The convergence tolerance of IBU is set to $10^{-4}$. The iteration number and pruning threshold of QuFEM are set to 2 and $10^{-4}$. The window size of DyREM is set to 4 to achieve a high data reuse rate.

**Evaluation platform.** We perform error mitigation experiments on the 136-qubit Quafu [16] quantum platform, where the readout error rate is $4.5\%$. We utilize QuFEM [7] to perform qubit partition and mitigation matrix characterization, with the number of qubits in a group set to 2. We implement Mthree [5] and IBU [6] on NVIDIA A100 GPU (using `cuBLAS` v12.4 and `JAX-GPU` v0.4.13, respectively). QuFEM is a software-friendly approach, and we implement it on an AMD EPYC 9554 64-core CPU (using `NumPy` v1.24.3). We implement SpREM accelerator on the Xilinx U50 FPGA using HLS C++, operating at 300 MHz.

### *B. Evaluation of Hardware Performance*

Table I shows the average hardware performance comparison of DyREM with baselines, evaluated on the VQE, QAOA, and DJ algorithms, using 16, 20, 24, and 28 qubits.

**Latency:** DyREM achieves a geometric average speedup of $9.6\times \sim 2000\times$ over baselines. Such a high speedup compared to conventional computing platforms (CPU and GPU) is attributed to the specialized computational architecture, which effectively detects and avoids redundant operations while generating the mitigation matrix. Although SpREM [10] exploits the Hamming sparsity to avoid operations involving zero values, it fails to eliminate redundant computations based on the noisy distribution. In contrast, DyREM compresses the mitigation matrix based on the output sparsity and detects redundant operations, achieving a maximum speedup of $73.8\times$ in the VQE algorithm compared to SpREM.

**Q-throughput:** We define q-throughput, measured in states per second (states/s), as a metric to evaluate mitigation performance. The geometric average q-throughput of DyREM for VQE, QAOA, and DJ algorithms reaches $4.2 \times 10^6$ states/s, $3.5 \times 10^6$ states/s, and $5.3 \times 10^6$ states/s, representing an improvement over baselines of $2.4\times \sim 2726\times$, $2.3\times \sim 2420\times$, and $1.5\times \sim 2002\times$, respectively. In particular, DyREM achieves $2.4\times$, $2.3\times$, and $1.5\times$ mitigation performance improvements over the custom accelerator SpREM [10]. Rather than relying on external memory to obtain the mitigation matrix (i.e., SpREM), DyREM utilizes the downsampled matrices to calculate the mitigation matrix on-chip. This strategy

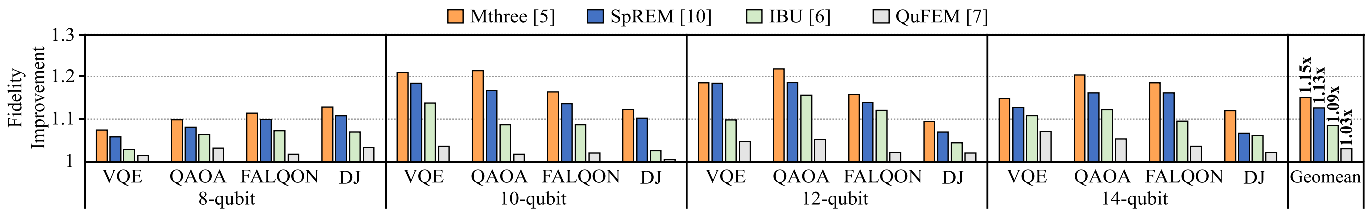


Fig. 7. Fidelity improvement over Mthree [5], SpREM [10], IBU [6], and QuFEM [7] on the benchmarks.

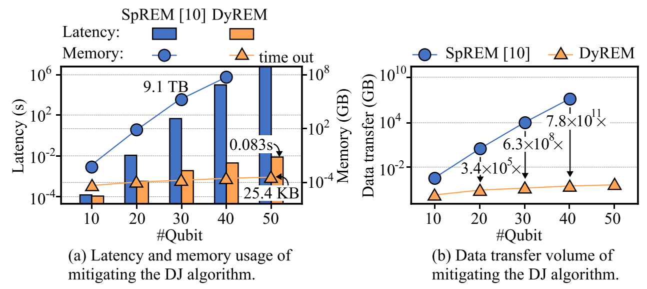


Fig. 8. Comparison between SpREM [10] and DyREM on the DJ algorithm.

effectively reduces the bandwidth demand and improves the utilization of compute units, thereby enhancing throughput.

## C. *Evaluation of Fidelity*

Although DyREM demonstrates outstanding hardware performance, it does not compromise on mitigation accuracy. Figure 7 illustrates the fidelity improvement of DyREM over the baselines. DyREM achieves an average fidelity improvement of 1.15×, 1.13×, 1.09×, and 1.03× over Mthree [5], SpREM [10], IBU [6], and QuFEM [7], respectively. The improvement comes from the fact that DyREM confines the mitigation space and utilizes the group matrices that consider the crosstalk. Conversely, Mthree and IBU use $2 \times 2$ mitigation matrices to construct the full-dimensional tensored mitigation matrix, failing to consider crosstalk between qubits. SpREM sets a large number of elements in the mitigation matrix to zero using Hamming distance, introducing system errors and reducing mitigation accuracy. Compared to QuFEM, DyREM confines the mitigation within the noise space, facilitating the convergence of noise probabilities towards ideal probabilities.

## D. *Comparison with SpREM*

Figure 8 presents the mitigation performance comparison between SpREM [10] and DyREM on the DJ algorithm, with the number of qubits ranging from 10 to 50.

**Latency and memory usage.** Figure 8(a) shows the latency and memory usage comparison between SpREM [10] and DyREM. Based on nonzero state-oriented computation, DyREM achieves linear time and space complexity. For instance, in the 50-qubit DJ algorithm, DyREM requires only 0.083 seconds of latency and 25.4 KB of memory, whereas SpREM fails to output the mitigation results. The scalability of DyREM stems from the on-chip generation of the mitigation matrix, which circumvents the limitations imposed by finite bandwidth. In contrast, SpREM relies on reading the mitigation matrix from off-chip memory, where the matrix size grows exponentially with the number of qubits, making bandwidth a bottleneck that limits its fast mitigation.

**Data transfer.** Figure 8(b) compares the data transfer volume between SpREM [10] and DyREM. The data transfer volume of DyREM is linearly increased with the number of qubits, from 0.63 KB (10-qubit) to 37.8 KB (50-qubit), while SpREM exhibits exponential complexity. This is because DyREM only needs to retrieve the measured qubits and noisy distribution from off-chip memory and calculate the mitigation matrix on-chip. Conversely, the mitigation matrix for SpREM is too large to store on-chip (e.g., 9.1 TB for 30-qubit), leading to increased data transfer volume.

## VI. Related Work

**Matrix-based readout error mitigation.** Mthree [5] and SpREM [10] exploit Hamming distance to compress the mitigation matrix. To implement scalable mitigation, IBU [6] utilizes a series of $2 \times 2$ qubit-independent sub-mitigation matrices to construct the mitigation matrix. To improve fidelity, CTMP [17] leverages Markovian noise models to mitigate correlated crosstalk errors. QuFEM [7] formulates the mitigation process as a tensor product using grouping matrices.

**Other error mitigation techniques.** Jigsaw [9] executes quantum programs in two models and employs a Bayesian post-processing step to mitigate measurement errors. HAMMER [18] combines the structure of quantum errors with a reclassification protocol to improve fidelity. Q-BEEP [19] introduces a Hamming spectrum model for characterizing localized and distant clustered Hamming errors. QuTracer [1] traces the states of qubit subsets to mitigate both gate and measurement errors.

## VII. Conclusion

This paper proposes DyREM, a co-design approach that dynamically mitigates readout error with an embedded accelerator. The key innovation lies in exploiting the inherent sparsity in the nonzero probability distribution of quantum states and calculating the tensor product on an embedded accelerator. To adapt to varying measured qubits, our dataflow dynamically downsamples the original mitigation matrix. We then design DyREM architecture that flexibly gates the redundant computation of nonzero quantum states. DyREM achieves remarkable improvements compared to other readout mitigation methods.

## Acknowledgment

This work was supported by National Key Research and Development Program of China (No. 2023YFF0905200) and Zhejiang Provincial Natural Science Foundation of China under Grant (No.LR25F020002).

## REFERENCES

[1] P. Li *et al.*, "Qutracer: Mitigating quantum gate and measurement errors by tracing subsets of qubits," in *ISCA*, 2024, pp. 103–117.

[2] S. Maurya *et al.*, "Scaling qubit readout with hardware efficient machine learning architectures," in *ISCA*, 2023.

[3] IBMQ. (2024) Ibm quantum platform. [Online]. Available: https://quantum.ibm.com/

[4] Rigetti. (2024) Rigetti computing: Quantum computing. [Online]. Available: https://www.rigetti.com/

[5] P. D. Nation *et al.*, "Scalable mitigation of measurement errors on quantum computers," *PRX Quantum*, vol. 2, p. 040326, 2021.

[6] B. Pokharel *et al.*, "Scalable measurement error mitigation via iterative bayesian unfolding," *Phys. Rev. Res.*, vol. 6, p. 013187, 2024.

[7] S. Tan *et al.*, "Qufem: Fast and accurate quantum readout calibration using the finite element method," in *ASPLOS*, 2024, p. 948–963.

[8] B. Lienhard *et al.*, "Deep-neural-network discrimination of multiplexed superconducting-qubit states," *Physical Review Applied*, vol. 17, no. 1, p. 014024, 2022.

[9] P. Das *et al.*, "Jigsaw: Boosting fidelity of nisq programs via measurement subsetting," in *MICRO*, 2021, p. 937–949.

[10] H. Zhang *et al.*, "Sprem: Exploiting hamming sparsity for fast quantum readout error mitigation," in *DAC*, 2024.

[11] S. Atchley *et al.*, "Frontier: exploring exascale," in *SC*, 2023, pp. 1–16.

[12] A. Kandala *et al.*, "Hardware-efficient variational quantum eigensolver for small molecules and quantum magnets," *nature*, vol. 549, no. 7671, pp. 242–246, 2017.

[13] E. Farhi *et al.*, "A quantum approximate optimization algorithm," *arXiv preprint arXiv:1411.4028*, 2014.

[14] D. Deutsch *et al.*, "Rapid solution of problems by quantum computation," *Proceedings of the Royal Society of London. Series A: Mathematical and Physical Sciences*, vol. 439, no. 1907, pp. 553–558, 1992.

[15] A. B. Magann *et al.*, "Feedback-based quantum optimization," *Physical Review Letters*, vol. 129, no. 25, p. 250502, 2022.

[16] B. A. of Quantum Information Sciences. (2022) Quafu quantum cloud computing platform. [Online]. Available: https://quafu.baqis.ac.cn/

[17] S. Bravyi *et al.*, "Mitigating measurement errors in multiqubit experiments," *Physical Review A*, vol. 103, no. 4, p. 042605, 2021.

[18] S. Tannu *et al.*, "Hammer: boosting fidelity of noisy quantum circuits by exploiting hamming behavior of erroneous outcomes," in *ASPLOS*, 2022, pp. 529–540.

[19] S. Stein *et al.*, "Q-beep: Quantum bayesian error mitigation employing poisson modeling over the hamming spectrum," in *ISCA*, 2023, pp. 1–13.