Vol: 3, Issue: 1, 2025
Pages:58-74
Received: 10 June 2025
Accepted: 24 June 2025
© 2025 Karabük University

CURRENT TRENDS IN COMPUTING
Research Article
DOI: 10.71074/CTC.1716528

MATGEN: A REALISTIC SPARSE MATRIX GENERATOR USING SIGNAL PROCESSING AND IMAGE PROCESSING METHODS

ALI EMRE PAMUK[1], FARUK KAPLAN[1], YOUSIF SUHAIL[1], MERT ALTEKIN[1] AND FAHREDDIN SUKRU TORUN [1*]

[1] *Computer Engineering Department, Ankara Yıldırım Beyazıt University, 06020, Ankara, Türkiye*

ABSTRACT. **The limited size of publicly available sparse matrix datasets creates a significant challenge for benchmarking, testing, and validating algorithms in scientific computing, artificial intelligence and other data-intensive applications. Existing approaches such as random matrix generators or general data augmentation methods often fail to produce structurally realistic matrices. To address this gap, we present MatGen which a tool for generating realistic variations of a given sparse matrix using signal processing and image processing techniques. MatGen takes a real sparse matrix as input and produces structurally consistent matrices at different sizes, introducing controlled variation while preserving key sparsity patterns. We evaluate the effectiveness of MatGen by analyzing structural features and visual similarities between original and generated matrices. Experimental results show that MatGen can produce realistic, scalable sparse matrices suitable for a wide range of applications including benchmarking computational methods, and sparse data techniques.**

1. INTRODUCTION

Machine learning (ML) and deep learning (DL) models need wide range of data to capture complex patterns. This need is more important in sparse data settings since missing values and weak relations reduce model performance more [1]. Sparse matrices arise in many domains, such as recommendation systems [2], natural language processing (NLP), and scientific simulations. In these domains, the distribution of nonzero values often reflects important relationships or physical structures. However, only a few real sparse datasets are publicly available [3, 4]. This makes it difficult to train and test ML or DL models that use sparse data as input. Therefore, models may perform poorly when trained on limited data that does not reflect real structures.

In the literature, to solve this issue researchers apply data augmentation, transfer learning, or matrix completion techniques. Synthetic and random data generation is also used to enlarge datasets and improve model robustness. However, these methods often generate matrices that do not reflect realistic or

E-mail address: aliemrepamuk0@gmail.com, farukkaplan52@gmail.com, yousifhazardous@gmail.com, altekinmert.ceng@gmail.com, fstorun@aybu.edu.tr(*).

Key words and phrases. Sparse matrices, matrix generator, matrix scaling, data augmentation, signal processing, image processing.

structured patterns. Despite these efforts, creating new real sparse matrices from scratch can be challenging due to the need for domain-specific knowledge, simulation software, and complex preprocessing workflows. These requirements make large-scale generation infeasible for many users. Furthermore, matrices produced from physical simulations are usually computationally expensive to obtain and are not always publicly available.

To provide a more effective solution, we propose *MatGen*[1] which is a tool that generates realistic sparse matrices from real examples. MatGen can keep important structural properties present in real-world data and also allows controlled variation and size flexibility. It uses image processing and signal processing techniques to capture sparsity and structure effectively. Controlled randomness adds diversity without losing important structural features.

In the experiments, we compare generated matrices with their original counterparts. We evaluate similarity using density, bandwidth, profile, row and column-wise distribution statistics, and other sparsity related features. Our results show that MatGen produces realistic and structurally consistent sparse matrices. The generated matrices can preserve key structural characteristics while allowing variation in size and sparsity. We also analyze visual patterns to measure how well the methods maintain the shape and distribution of nonzero entries.

The main scientific contributions of this work are threefold. First, we design and implement a new framework called `MatGen`, which can generate realistic sparse matrices by transforming existing matrices. Second, we adapt a variety of scaling techniques originally developed for image and signal processing. These adaptations allow the techniques to work effectively with sparse matrices while preserving key structural features. Third, we conduct detailed experiments to evaluate the quality of the generated matrices. We compare them using visual patterns and structural features across different matrix sizes and transformation types. This work is especially important for scientific computing, numerical simulation, and machine learning. In these areas, realistic sparse matrices may be required for tasks such as benchmarking, evaluating linear solvers, and developing algorithms that use matrix structure.

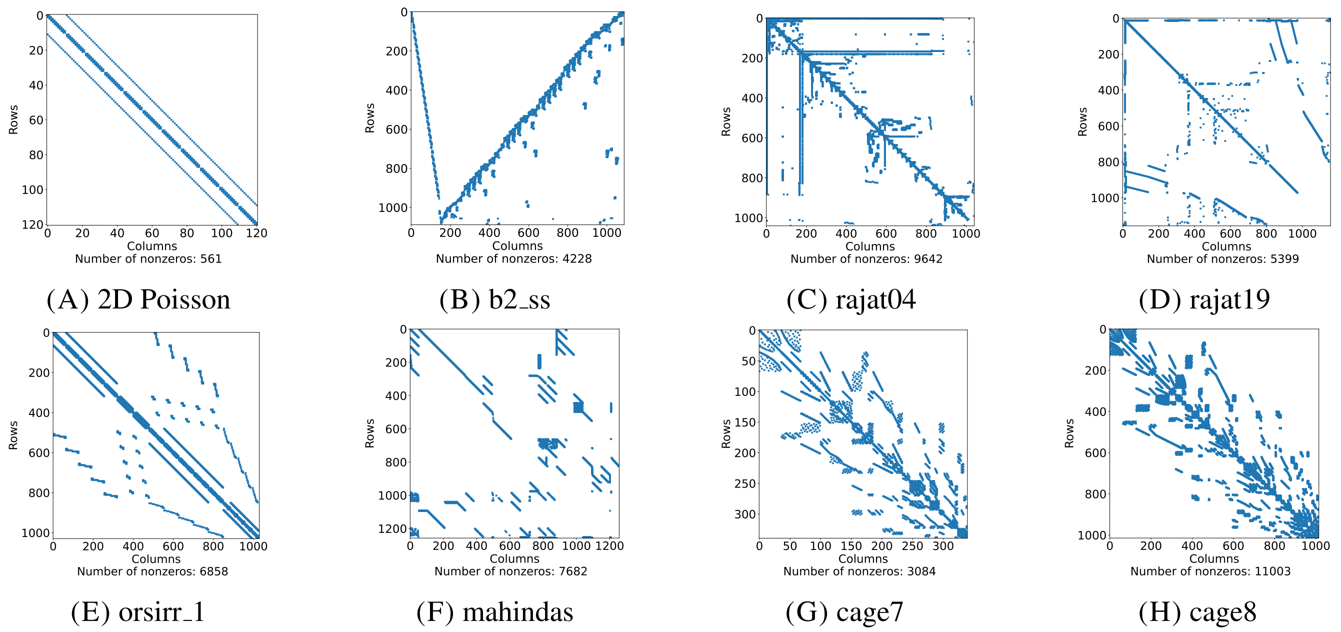
2. Sparse Matrices: Structure and Applications

Sparse matrices appear in many areas like science, engineering, AI, and medical imaging. Using sparse data structures for them, we can save memory and speed up computation. This is useful in large problems such as optimization and solving linear systems. Such problems often come from simulations, data models, or high-performance computing application domains.

The structural properties of sparse matrices can vary considerably across different application domains. For instance, matrices resulting from PDE-based physical simulations often show structured sparsity, such as banded or block-diagonal patterns. On the other hand, those from network analyses, chemical simulation or data-driven models may show irregular or clustered patterns. Figure 1 shows the sparsity patterns from a variety of applications, demonstrating the diversity in matrix structure.

Figure 1A shows a classic example which is a banded sparse matrix that results from the discretization of the 2D Poisson equation using finite differences. Other figures show real-world sparse matrices from

[1]https://github.com/AYBU-ParLab/MatGen



(A) 2D Poisson (B) b2_ss (C) rajat04 (D) rajat19

(E) orsirr_1 (F) mahindas (G) cage7 (H) cage8

FIGURE 1. **Comparison of different sparsity patterns.**

various domains. These include chemical process simulation (Figure 1B), circuit simulation (Figures 1C and 1D), computational fluid dynamics (Figure 1E), an economic model (Figure 1F), and electrophoresis (Figures 1G and 1H). Understanding and utilizing the properties of these various matrices are important for designing efficient algorithms for tasks such as sparse matrix-vector multiplication, iterative solvers, preconditioning techniques and AI or ML applications involving sparse data.

Within specific scientific domains and problem types, sparse matrices often share consistent structural characteristics. These similarities typically arise from common discretization methods, geometric constraints, or data organization. For instance, matrices resulting from finite element discretizations of PDEs usually have structured sparsity related to the mesh. On the other hand, matrices from social networks or recommendation systems often reflect local interactions and exhibit irregular patterns.

Figure 1 also highlights the structural similarities that often occur within the same application domain. For instance, the matrices from circuit simulation (Figures 1C and 1D) and electrophoresis problems (Figures 1G and 1H) show closely related sparsity patterns. These examples show that some structural properties often repeat in matrices from similar application areas.

Sparse matrices also play an important role in machine learning and deep learning. In recommender systems [2], user-item interaction matrices are typically sparse because users engage with only a small fraction of the available items. In natural language processing (NLP) [5], term-document matrices and word embeddings are sparse due to the limited co-occurrence of words in large corpora. These matrices are commonly stored using compressed formats and often processed using matrix factorization techniques such as singular value decomposition (SVD), non-negative matrix factorization (NMF) or principal component analysis (PCA). In seismic imaging [6] and other scientific fields [7–9], sparse matrix structures derived from similar models are also exploited for compression, memory usage economy, and computational acceleration.

In deep learning, sparsity is utilized to improve performance through pruning [10], leading to sparse weight matrices that reduce memory and computation without compromising accuracy. Graph neural networks (GNNs) [11,12] operate directly on sparse adjacency matrices to capture complex relationships in structured data, such as social networks or molecular graphs.

3. Related Work

Although sparse matrices are widely used in machine learning, statistics, and data science, there is limited access to real-world examples. Several resources and tools have been developed to support the study and application of sparse data. Among the most widely used is the SuiteSparse Matrix Collection (formerly the UF Sparse Matrix Collection) [3], which provides access to a broad set of sparse matrices from real applications, including structural analysis, circuit simulation, and optimization. While this collection is essential for benchmarking, it consists of a fixed set of problem-specific matrices and does not support the generation of new matrices with user-defined structural properties or varying dimensions.

Due to the limited availability of real-world matrices, researchers often rely on synthetic data generators. Standard tools such as scipy.sparse.random in Python and the sprand function in MATLAB allow users to create sparse matrices with a specified density and size. However, the sparsity patterns generated by these tools are typically unstructured and uniformly random. These patterns are often unrealistic and fail to capture the complex structures and dependencies observed in matrices derived from physical simulations or real-world data sources.

In some application-specific domains, matrix generation is guided by graph models. The R-MAT model [13] and Stochastic Block Models [14] are widely used to generate synthetic adjacency matrices with community structure. These models are useful for simulating social and biological networks, but they are not intended for general sparse matrix generation, especially where solver-relevant properties are important.

Alternative methods uses fixed templates or structural rules to generate matrices with specific patterns. For instance, Farhadi et al. [15] propose an approach for creating structured sparse matrices based on a predefined format. However, such methods are limited in flexibility and do not adapt to the structure of existing data. They also lack mechanisms for learning or transferring patterns from real examples to new, synthetic instances.

Recent developments in ML and AI highlight the importance of realistic sparse matrix generation. In graph neural networks (GNNs), sparse matrix–matrix multiplication is a core operation, and its performance heavily depends on the input sparsity structure [16]. Similarly, sparse attention mechanisms used in transformer architectures rely on structured sparsity to reduce memory and computation costs [17]. These applications underscore the importance of developing tools capable of generating structured and realistic sparse matrices.

Despite recent developments in synthetic data generation, there is no general-purpose tool that currently exists for generating sparse matrices by capturing structural patterns from real-world matrices across scientific or engineering domains. To address this gap, we propose MatGen, a tool that extracts structural features from a real sparse matrix and uses signal processing and image processing techniques to generate new matrices that preserve these features. This approach enables the creation of datasets with realistic variability while maintaining the complexity and essential structural properties of the original data.

4. Proposed Methods

In this section, we present our methods in MatGen for generating realistic sparse matrices. The MatGen includes eight approaches which are based on Nearest Neighbor Interpolation, Bilinear Interpolation, Image-based Rescaling, Lanczos Interpolation, Gaussian Pyramid, Discrete Fourier Transform, Discrete Cosine Transform, and Wavelet Transform techniques. Among these, Gaussian Pyramid-based method is designed specifically for downscaling, while the others support both upscaling and downscaling.

4.1. **Nearest-Neighbor Interpolation.**

Nearest-neighbor interpolation [18] is a well-established technique which is originally developed for applications in image processing and computer graphics. It assigns to each interpolated point the value of its closest known neighbor, resulting in a piecewise-constant approximation. In MatGen, this method is adapted to scale sparse matrices while preserving their sparsity structure. It maps the coordinates of existing nonzero entries to new positions.

For a coordinate (x,y), the interpolated value is:

$$f(x,y) = f(\text{round}(x/s), \text{round}(y/s)), \tag{1}$$

where s is the scaling factor. During upscaling, this method replicates nearby values that lead to increasing the number of nonzeros while maintaining the original pattern. Since it avoids dense 2D matrix conversion and preserves nonzero distribution well, it is especially useful in scenarios where structural patterns is more important than maintaining exact sparsity.

4.2. **Bilinear Interpolation.**

In MatGen, we adapt Bilinear interpolation to generate scaled sparse matrices. Bilinear interpolation computes an interpolated value at a given continuous location (x,y) based on the four nearest points in a grid. Formally, let (x,y) be a location within a rectangular region defined by points (x_i,y_j), (x_{i+1},y_j), (x_i,y_{j+1}), and (x_{i+1},y_{j+1}). The interpolated value $f(x,y)$ is given by:

$$f(x,y) = f(x_i,y_j)(1-\alpha)(1-\beta) + f(x_{i+1},y_j)\alpha(1-\beta) + f(x_i,y_{j+1})(1-\alpha)\beta + f(x_{i+1},y_{j+1})\alpha\beta, \tag{2}$$

where $\alpha = x - x_i$ and $\beta = y - y_j$. Although Bilinear interpolation usually smooths values, in our sparse setting it tends to produce smaller interpolated values near zero, which are then removed by filtering operation. This results in output matrices that are often sparser than the original.

4.3. **Image-Based Rescaling.** Sparse matrices can be naturally represented as images, where nonzero entries becomes corresponding pixels in the image. Thus, image processing techniques become powerful tools for capturing and generating structural patterns. However, since images are stored in 2D dense format keeping them is very expensive for very large matrices. Therefore, they should be used carefully.

In Matgen, our Image-based rescaling method begins by converting the matrix into a visual heatmap, where nonzero values are color-coded and zero entries appear as white. This heatmap is then converted to a grayscale image, where pixel values corresponds to the magnitude of matrix entries. The image is resized to the desired dimensions using a resampling method. In our setting, we use the box resampling method [19], which is efficient and suitable for preserving average intensity. Furthermorer, other interpolation techniques such as bilinear or bicubic resampling can also be applied. After resizing, the image is mapped back to a sparse matrix. This approach provides a simple and general mechanism to approximate rescaled versions of a matrix.

4.4. **Gaussian Pyramid Downscaling.** The Gaussian pyramid method [20] is a classical image processing technique that smooths and subsamples a matrix to produce lower-resolution versions. In our approach, the Gaussian pyramid method is used to successively smooth and subsample the original matrix. Given a sparse matrix A, the downscaled version A_{reduced} is obtained by applying a Gaussian filter G and subsampling:

$$A_{\text{reduced}}(i,j) = \sum_m \sum_n A(2i+m, 2j+n)\, G(m,n), \tag{3}$$

where $G(m,n)$ is a Gaussian weighting kernel defined as:

$$G(m,n) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{m^2+n^2}{2\sigma^2}\right). \tag{4}$$

Gaussian pyramid downscaling preserves important structural features of the original sparse matrix. It captures sparsity patterns at multiple levels of resolution. However, due to its fixed halving behavior, it is only applicable when generating significantly smaller matrices, such as those with half the number of rows and columns. This method does not support resizing to intermediate or closely related dimensions.

4.5. **Lanczos Resampling.** In MatGen, we propose a Lanczos resampling–based method [21] for sparse matrix scaling, adapted from classical interpolation techniques in signal processing. The method uses a sinc-based kernel to compute new values based on nearby entries in the matrix. It selects values using both direct mapping from the original nonzeros and random sampling to cover the whole matrix. To keep the matrix sparse, only values above a certain threshold are kept.

Lanczos Resampling involves kernel-based resampling which could preserve structural details better than simple interpolation methods. Given a continuous coordinate x, the Lanczos kernel $L_a(x)$ with parameter a is defined as:

$$L_a(x) = \begin{cases} \operatorname{sinc}(x) \operatorname{sinc}\left(\frac{x}{a}\right), & \text{if } -a < x < a, \\ 0, & \text{otherwise,} \end{cases} \tag{5}$$

where the sinc function is:

$$\operatorname{sinc}(x) = \frac{\sin(\pi x)}{\pi x}. \tag{6}$$

The interpolated value $f(x,y)$ using Lanczos kernels is computed as:

$$f(x,y) = \sum_i \sum_j f(i,j) L_a(x-i) L_a(y-j). \tag{7}$$

Lanczos scaling preserves sharp transitions and important sparsity patterns. This helps generate sparse matrices similar to the original matrix in structure and properties.

4.6. **Discrete Fourier Transform (DFT).** The Discrete Fourier Transform (DFT) [22] is a classical tool in signal processing that represents data as a sum of sinusoidal components. In our approache, we use DFT method to resizes a sparse matrix by operating in the frequency domain. It uses the two-dimensional DFT, which expresses the matrix as a sum of sinusoidal components. This formulation captures global patterns in the matrix. This enables size changes through spectral manipulation.

To apply the method, the matrix is first transformed using the 2D Fast Fourier Transform (FFT). This process produces a frequency-domain representation of the matrix, denoted by $F(u,v)$, and is defined as follows:

$$F(u,v) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x,y) e^{-2\pi i \left(\frac{ux}{M} + \frac{vy}{N}\right)}$$

In the frequency domain, the spectrum is shifted so that low-frequency components are centered. The frequency matrix is then either cropped or zero-padded around the center to match the desired output size. After adjusting the spectrum, the method applies the inverse FFT to return to the spatial domain and reconstruct the scaled matrix. Finally, small values below a threshold are set to zero to recover a sparse representation.

This method requires converting the matrix to a dense format and performing full FFT operations. Similarly, the image-based rescaling method also relies on dense matrix representations for resizing. Therefore, the cost of memory and computation can be high for both approaches if the sparse matrix is large.

4.7. **Discrete Cosine Transform (DCT).** In MatGen, the Discrete Cosine Transform (DCT) [23] method is used to resize sparse matrices by operating in the frequency domain, like DFT method. While both DFT and DCT operate in the frequency domain, DCT focuses on low-frequency components. This makes it more suitable for capturing important structural information with fewer coefficients.

Given a matrix $M \in \mathbb{R}^{n \times n}$, the two-dimensional DCT is defined as:

$$M_{u,v} = \alpha(u)\alpha(v) \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} M_{i,j} \cos\left(\frac{\pi(2i+1)u}{2n}\right) \cos\left(\frac{\pi(2j+1)v}{2n}\right), \tag{8}$$

where

$$\alpha(k) = \begin{cases} \sqrt{1/n}, & \text{if } k = 0, \\ \sqrt{2/n}, & \text{otherwise.} \end{cases}$$

To resize a sparse matrix, first the DCT method is applied to the original matrix. The coefficient matrix is resized by zero-padding or interpolation in the frequency domain, followed by the inverse DCT:

$$M_{i,j} = \sum_{u=0}^{n-1}\sum_{v=0}^{n-1} \alpha(u)\alpha(v)M_{u,v}\cos\left(\frac{\pi(2i+1)u}{2n}\right)\cos\left(\frac{\pi(2j+1)v}{2n}\right). \tag{9}$$

Values below a threshold are dropped to maintain the sparsity of the upscaled matrix. This step ensures that the result remains efficient in memory and computation while preserving important structure. The full DCT has memory complexity $O(n^2)$ and computation complexity $O(n^2 \log n)$. When applied block-wise, the memory complexity is reduced to $O(\text{nnz})$, and computation becomes $O(n^2 \log b)$, where b is the block size.

4.8. **Wavelet Transform.** In MatGen, the Wavelet Transform method is used to rescale sparse matrices using multi-level 2D wavelet transforms. The matrix is partitioned into blocks, and each block is transformed to get low-frequency and high-frequency coefficients. These coefficients are resized and the block is reconstructed using the inverse Wavelet transform.

In wavelet transform, the block is repeatedly split into approximation and horizontal, vertical, and diagonal components. The number of decomposition levels controls how many times this splitting is applied. Higher levels provide a finer multiscale representation of the data. This approach allows for localized, resolution-aware approximation of the original matrix while preserving sparsity.

5. Experimental Results and Discussion

We analyze the performance of the eight methods on a set of real-world sparse matrices from the SuiteSparse matrix collection [3]. Four versions of each original matrix are generated with varying size. A double-size version, matrix with double the original size, is created to test how well each method handles interpolation and structural consistency during upscaling. A half-size version is generated to assess how effectively the methods preserve dominant patterns and sparsity during compression or downsizing. Additionally, we generate two slightly modified versions of the matrices. One with size expanded by one $(N+1)$ and another reduced by one $(N-1)$ are included to examine the methods performance.

To measure how well each method keeps the structure of the matrix after resizing, we use cosine similarity (10) between the structural feature vectors of the original and scaled matrices. Given two feature vectors u and v, the cosine similarity is calculated as:

$$\text{cosine\_similarity}(u,v) = \frac{u \cdot v}{||u|| \cdot ||v||} \tag{10}$$

where $u \cdot v$ is the dot product, and $||u||$ and $||v||$ are the Euclidean norms of the vectors. These feature vectors describe various aspects of sparsity and shape of the matrices.

In Table 3, provided in Appendix, the actual feature values that are used in similarity calculation are presented for a sample matrix (`cavity03`) in our dataset. In similarity calculation we used normalized values of the features with respect to size of matrices. The average distance to the diagonal, bandwidth, and profile capture how entries are spread around the main diagonal. Density represents the ratio of nonzeros to the matrix size. Row-wise and column-wise maximum indices and standard deviations reflect value distribution along axes. The feature pattern symmetry measures how symmetric the matrix structure is. We also include the number of diagonals that contain nonzero values and the count of entries that break structural symmetry.

Table 1 shows the cosine similarity values for each method by taking average of cosine similarity results of all matrices in the dataset. Higher cosine similarity values can indicate better preservation of the matrix's structural properties. When the matrix is slightly enlarged by one row and column (Expand (+1)), the nearest neighbor method gives the best similarity (0.999), followed by the wavelet method (0.991). The Bilinear and DFT methods scores the lowest (0.637 and 0.613) similarity values. In the Upscale (2x) operation, Nearest neighbor (0.918) and Wavelet (0.906) perform better than others, while DFT (0.593) and Bilinear (0.615) perform worst.

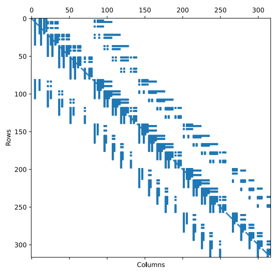
TABLE 1. **Average structural similarity (cosine) values across the dataset for various matrix generation methods**

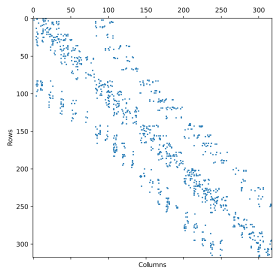
Method	Expand (+1)	Upscale (2x)	Reduce (-1)	Downscale (1/2x)
Bilinear	0.637	0.615	0.605	0.640
DCT	**0.976**	0.817	**0.974**	0.847
DFT	0.613	0.593	0.662	0.643
Image-based	0.805	0.750	0.805	0.786
Lanczos	**0.906**	0.825	0.893	0.800
NN	**0.999**	**0.918**	**0.999**	**0.945**
Wavelet	**0.991**	**0.906**	**0.988**	0.891
Gaussian	-	-	-	0.748

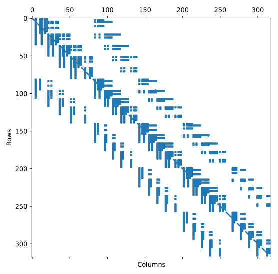
When the matrix is reduced to half by 1/2x downsizing operation, the Nearest neighbor (0.945) and Wavelet (0.891) methods are more successful in keeping the structure, while DFT and Bilinear again show lower cosine values. When the size is slightly reduced (Reduce (-1)), Nearest neighbor ans Wavelet again performs best (0.999 and 0.988), while Bilinear and DFT give the lowest results (0.605 and 0.662).

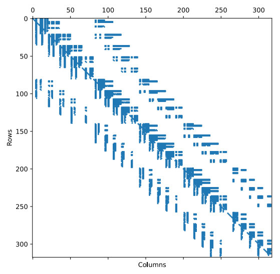
As seen in Table 1, for small changes (Expand and Reduce operations), the nearest neighbor method gives the best results, with cosine similarity above 0.99. Wavelet and DCT-based methods also perform very well in these cases. This shows that these methods can handle small changes without losing important structures in the generated matrices.

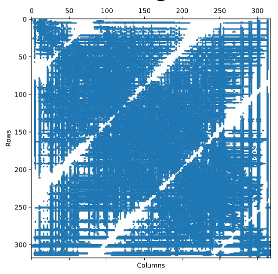
Figures 2, 3, and 4 show the sparsity patterns of the original matrix `cavity03` along with the scaled matrices generated by different methods using the Expand (+1), Upscaling 2x, Reduce (-1) and Downscaling 1/2x operations. Here, the original matrix is an unsymmetric sparse matrix from a finite element discretization of a computational fluid dynamics (CFD) problem [3]. It has a size of 317×317 and contains 7311 nonzero entries.

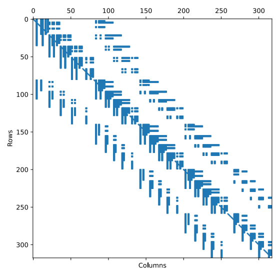

(A) Original

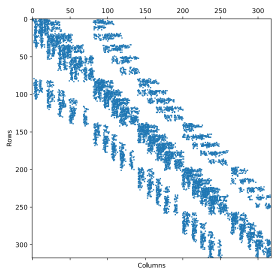

(B) Bilinear


(C) NN


(D) DCT


(E) DFT


(F) Image


(G) Lanczos

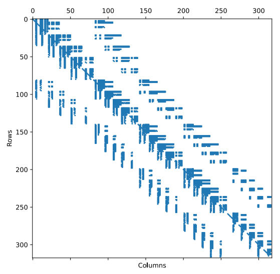

(H) Wavelet

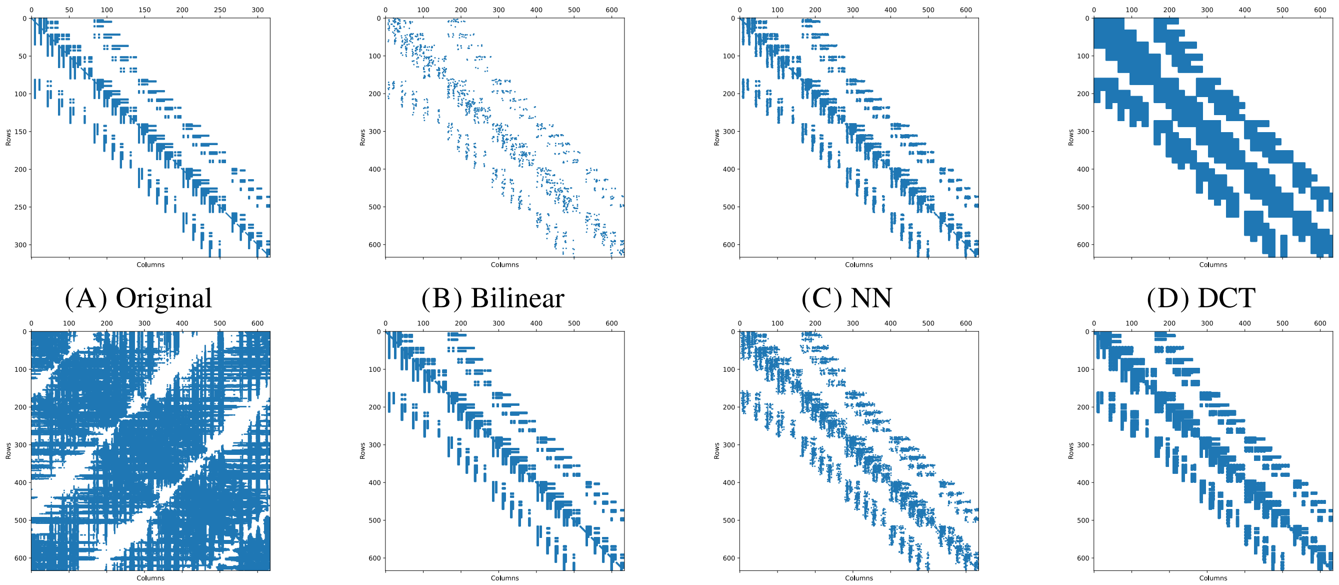
FIGURE 2. **Original and generated `cavity03` patterns after Expand (+1) operation.**

In Figure 2, all generated matrices (Figures 2B - 2H) have one additional row and column, resulting in a size of 318×318. However, the number of nonzero entries varies depending on the method. Among the generated matrices, the nearest neighbor (Figure 2C), DCT (Figure 2D), Image-based (Figure 2F) and Wavelet (Figure 2H) methods most closely preserve the structure of the original matrix. The main diagonal band and nearby nonzero locations are well preserved in the generated matrices of these methods. There is very little distortion in these results. This matches their high cosine similarity scores that is above 0.97. Lanczos (Figure 2G) give a visually similar result to DCT, however there is small shifts in the position of nonzeros. Bilinear interpolation (Figure 2B) leads to greater loss of detail and reduces sparsity, especially near the boundaries. The DFT (Figure 2E) introduces significant noise that results in a matrix pattern that differs considerably from the original. This is also consistent with its low similarity score of 0.613 across the dataset. The output of the method can be improved by stronger thresholding or matrix based finetuning after the inverse transform.

In Figure 2, although Bilinear and Lanczos does not generate identical matrices with the original matrix, this difference is not a drawback of the proposed method. In fact, it can be seen as a way to generate a new matrix that still keeps key structural features of the original. It adds a level of controlled variation, similar to randomization, but in a more intelligent and structured manner. This capability presents one of the main contributions of our work which is to is generating realistic but structurally distinctive matrices.

Figure 3 shows the patterns of the original and generated matrices after applying the 2× Upscaling operation on `cavity03`. All generated matrices have size 634×634, but the number of nonzeros varies by method. Compared to the Expand (+1) operation, the structural differences become more visible for bigger upscaling. Nearest neighbor (Figure 3C) and Lanczos (Figure 3G) still maintain the original banded structure well. However, DCT (Figure 3D) and Wavelet (Figure 3H), show more visible distortion and block enlargement due to interpolation in the frequency domain.

Bilinear interpolation (Figure 3B) again results in a loss of sparsity. This is consistent with its low average cosine similarity score of 0.615 across the dataset. DFT (Figure 3E) produces even more severe distortions by introducing high-frequency noise. This disrupts the matrix's original structure. This outcome is also consistent with its cosine similarity score of 0.593 across the dataset. Nearest neighbor and Wavelet methods maintain the original pattern more effectively, with similarity scores of 0.918 and 0.906. While DCT and Wavelet methods perform well in smaller scale changes (Figure 2), their accuracy decreases with bigger scaling.



(A) Original (B) Bilinear (C) NN (D) DCT

(E) DFT (F) Image (G) Lanczos (H) Wavelet

FIGURE 3. **Original and generated `cavity03` patterns after 2× Upscaling operation.**

Figure 4 display the matrix patterns generated after applying the Reduce (–1) and Downscale (1/2×) operations. In both operations, the goal is to shrink the original matrix while maintaining its important structural properties. In Figures 4B-4H, the patterns of generated matrices are shown after using Reduce (-1) operation which involves a minimal size reduction. In the figures, the results are more stable. Nearest neighbor (4C), Wavelet (4H), and DCT (4D) again perform best with similarity values above 0.97. This can be seen visually that these methods maintain almost perfect alignment with the original pattern (A). Lanczos (4G) also generate similar matrix with some noises at the off-diagonal areas. Methods like Bilinear (4B), DFT (4E), and Image-based (4F) show more deviation. DFT introduce significant noise as seen in earlier experiments.

In Figures 4I-4P, the patterns of generated matrices are shown after using 1/2x downsizing operation. As seen in the figures, Nearest neighbor (Figure 4J) and Wavelet (Figure 4P) methods preserve the original banded pattern with minimal distortion. This is consistent with their high cosine similarity scores with 0.945, and 0.891, respectively. Lanczos (Figure 4O) and Wavelet also retain much of the structure, although slightly less accurately. In contrast, Bilinear (Figure 4I), DFT (Figure 4M) and Gaussian (Figure 4L) methods introduce more sparsity loss and noise. This is also reflected in their lower similarity scores ranging from 0.640 to 0.748. These findings show that both for moderate and slight downsizing operations, Nearest neighbor, Wavelet, and DCT methods consistently preserve structural similarity visually and analytically.

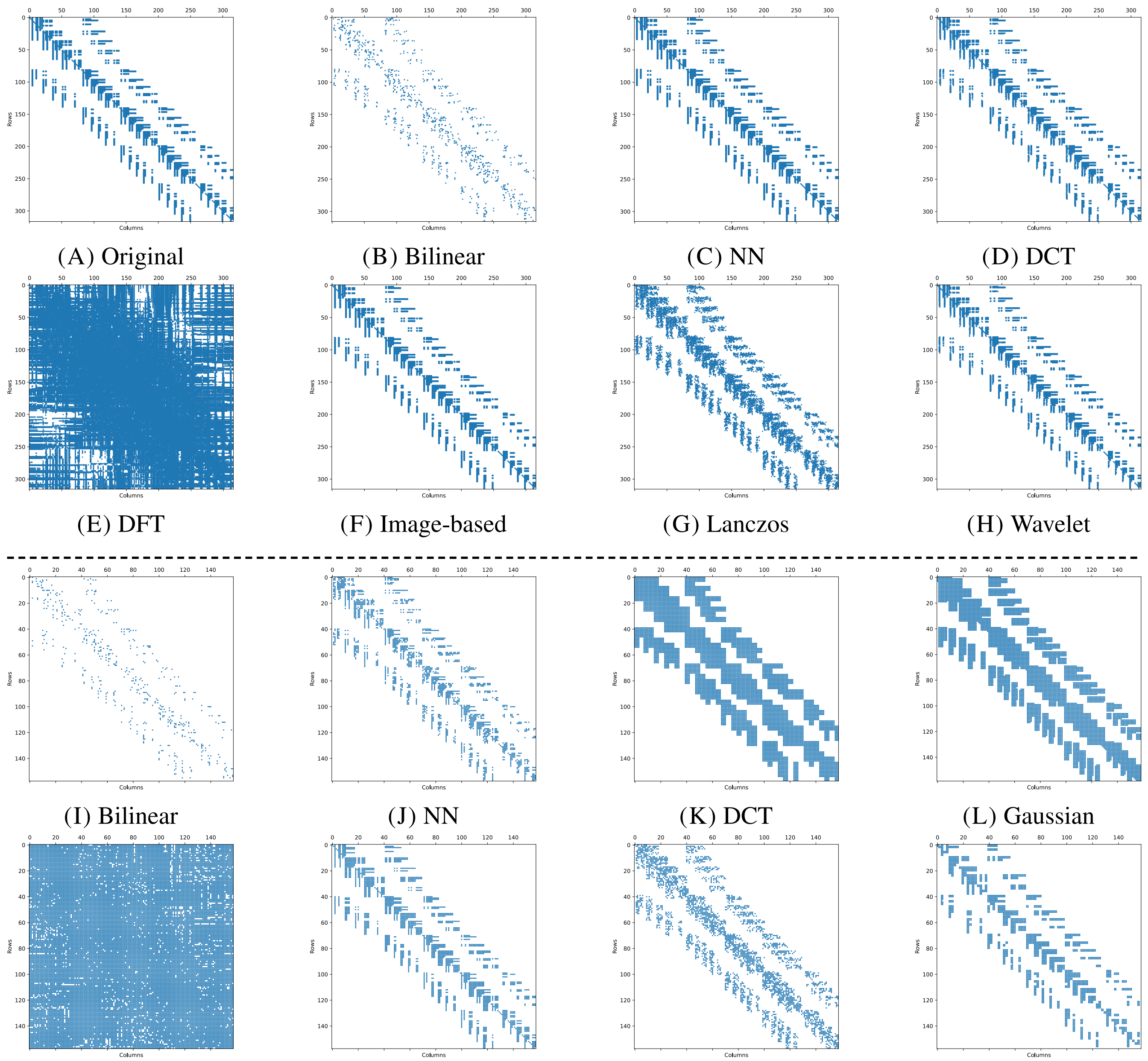
Among the evaluated techniques, Nearest Neighbor, Wavelet, and DCT methods consistently preserve the structural features of sparse matrices across both upscaling and downscaling operations. Nearest Neighbor retains original nonzero locations by duplicating them without smoothing, maintaining local sparsity. Wavelet transform captures localized features and low-frequency components, allowing it to preserve diagonal and block patterns under size changes. DCT compresses global structure into low-frequency coefficients, which helps it maintain the overall sparsity layout even when the matrix is resized.

Scalability and Large-Scale Matrix Generation. One of the important goals of `MatGen` is to generate large-scale sparse matrices that preserve the structural properties of real input matrices. To evaluate this, we test the scalability of our methods by applying $4\times$ upscaling operations to a large sparse matrix. For this experiment, we use the `bcsstk30` matrix from the SuiteSparse Matrix Collection. `bcsstk30` is a large symmetric sparse matrix with a dimension of $28{,}924 \times 28{,}924$ and contains 1,036,208 nonzero entries. It originates from a structural engineering problem and represent the stiffness matrix of an offshore generator platform.

In these experiments, we use methods that support upscaling without requiring dense matrix conversions. Specifically, we apply the Nearest-neighbor (NN), Lanczos, Wavelet, and DCT-based techniques, which can operate efficiently on large matrices. By using these methods, the 4x upscaling operation applied on `bcsstk30` to produce matrices of size $115{,}696 \times 115{,}696$.

Table 2 shows the details of the original matrix and the matrices generated by different upscaling methods. The column labeled *Sim* indicates the cosine similarity value for each method, while the other columns are explained in the Appendix Section. Among the methods, DCT and Wavelet show better performance in preserving the structural properties of the original matrix. In contrast, matrices generated using Lanczos and NN show a clear loss of structural similarity. It is also seen by their lower pattern symmetry (Psym) values.

Figure 5 shows visualizations of the original matrix and the generated matrices from `bcsstk30`. Due to the large size of these matrices, differences are not easily visible at this resolution. Although all images



(A) Original (B) Bilinear (C) NN (D) DCT

(E) DFT (F) Image-based (G) Lanczos (H) Wavelet

(I) Bilinear (J) NN (K) DCT (L) Gaussian

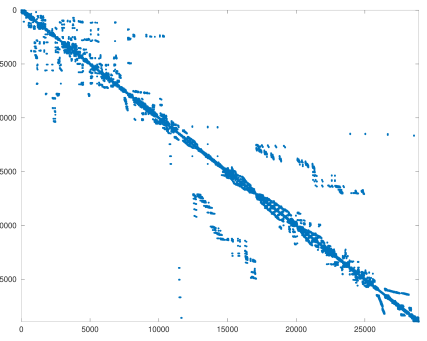
(M) DFT (N) Image-based (O) Lanczos (P) Wavelet

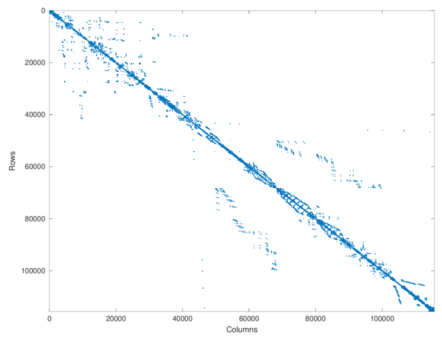
FIGURE 4. **Original and generated `cavity03` matrix patterns after Reduce (-1) (Figures (B-H)) and 1/2x Downscaling (Figures (I-P)) operations.**

TABLE 2. **Details of the `bcsstk30` and generated matrices from it for 4x upscaling experiment**

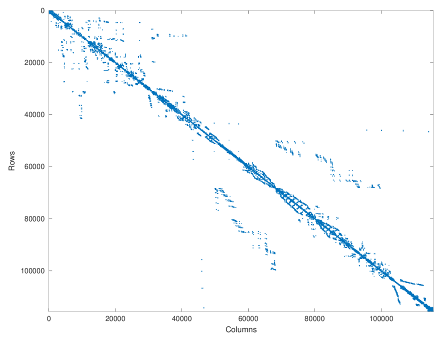
Method	Sim	N	NNZ	Density	Psym	Diag	Ndg	Dist	Band	Profile	Rmx	Rmi	Rstd
Original	1.00	28,924	2,043,492	0.0024	1.00	28924	6619	461	16947	41327	219	4	31.7
Lanczos	0.27	115,696	2,104,732	0.0002	0.33	29019	12865	1850	67807	506529	86	0	14.2
NN	0.42	115,696	7,331,809	0.0005	0.41	45887	26525	1844	67791	668956	321	0	40.5
Wavelet	0.77	115,696	41,627,296	0.0031	1.00	115696	28528	1938	67805	752373	1102	24	151.7
DCT	0.89	115,696	80,808,184	0.0060	1.00	115696	30422	2262	67839	919760	1664	160	239.7

appear similar, their structural characteristics vary significantly, as shown in Table 2. High-resolution versions of these figures are available in our software repository for more detailed comparison.

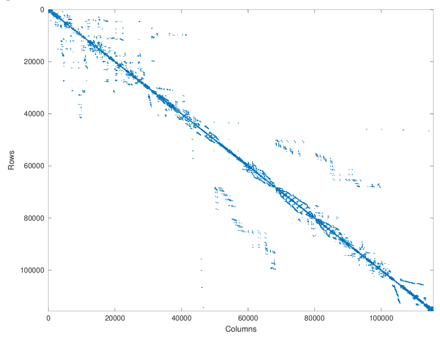

(A) Original



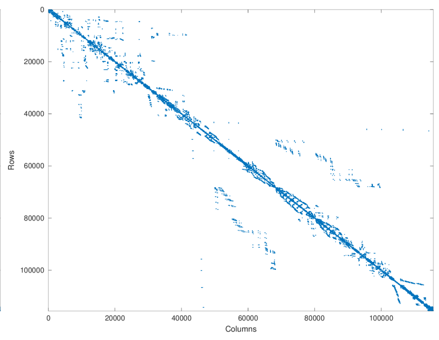
(B) NN



(C) DCT



(D) Lanczos



(E) Wavelet

FIGURE 5. Original and generated `bcsstk30` matrix patterns after 4× Upscaling operation.

These results show that MatGen can scale real matrices to much larger sizes while maintaining the visual and structural features of the original pattern. This demonstrates its suitability for generating synthetic test problems in large-scale simulations, benchmarking, and machine learning applications. An additional advantage of MatGen is that the generated matrices look visually similar to the original, even though their structural characteristics differ. This allows the creation of a wide range of test cases that preserve essential patterns but vary in complexity.

6. Conclusion and Future Work

In this work, we introduce MatGen which is a framework for generating realistic sparse matrices through scaling techniques while preserving important structural properties. These techniques include interpolation and transform-domain methods which are originated from image and signal processing area. We evaluate eight methods under four types of sparse matrix generation operations. The results show that Nearest neighbor, Wavelet, and DCT methods preserve the structural characteristics of the original matrices more effectively than the other methods evaluated in this study. These are supported by visualizing patterns of the generated matrices from a sample matrix. We also perform cosine similarity analysis on structural features to evaluate the methods.

MatGen is useful for areas like scientific computing, simulations, machine learning, signal processing, and image processing. These fields often need many realistic sparse matrices for benchmarking solvers, testing algorithms, and studying structural behavior under transformations. However, existing sparse matrix datasets are often small and limited. MatGen helps address this gap by generating structurally realistic and varied matrices for experimental evaluation.

As future work, we plan to examine the numerical values of the generated matrices. In this way, we ensure that MatGen produces not only structurally consistent but also numerically reliable sparse matrices. We also aim to extend the framework to rectangular matrices and explore adaptive strategies that choose the best scaling method based on matrix features.

Declarations

- **Conflict of Interest:** The authors declare that there is no conflict of interest regarding the publication of this work.
- **Data Availability:** The complete implementation code developed and used in this study, including all matrix transformation methods, evaluation procedures, and experimental configurations, is publicly accessible at the following repository: https://github.com/AYBU-ParLab/MatGen
- **Author Contributions:** Fahreddin Sukru Torun proposed the main idea, coordinated the research process, evaluated the methods and contributed to writing the paper. Ali Emre Pamuk, Faruk Kaplan and Mert Altekin implemented the methods, supported code development and contributed to the paper writing. Yousif Suhail carried out the experiments, resolved technical issues and software bugs, and contributed to the paper writing.

References

[1] I. Rish, G. Grabarnik, Sparse modeling: theory, algorithms, and applications, CRC press, 2014.

[2] T. M. A. U. Gunathilaka, P. D. Manage, J. Zhang, Y. Li, W. Kelly, Addressing sparse data challenges in recommendation systems: A systematic review of rating estimation using sparse rating data and profile enrichment techniques, Intelligent Systems with Applications (2025) 200474.

[3] T. A. Davis, Y. Hu, The university of florida sparse matrix collection, ACM Transactions on Mathematical Software (TOMS) 38 (1) (2011) 1–25.

[4] W. Hu, M. Fey, M. Zitnik, Y. Dong, H. Ren, B. Liu, M. Catasta, J. Leskovec, Open graph benchmark: Datasets for machine learning on graphs, Advances in neural information processing systems 33 (2020) 22118–22133.

[5] R. Drikvandi, O. Lawal, Sparse principal component analysis for natural language processing, Annals of data science 10 (1) (2023) 25–41.

[6] N. Liu, Y. Lei, R. Liu, Y. Yang, T. Wei, J. Gao, Sparse time–frequency analysis of seismic data: Sparse representation to unrolled optimization, IEEE Transactions on Geoscience and Remote Sensing 61 (2023) 1–10.

[7] F. S. Torun, M. Manguoglu, C. Aykanat, Parallel minimum norm solution of sparse block diagonal column overlapped underdetermined systems, ACM Trans. Math. Softw. 43 (4) (Jan. 2017). `doi:10.1145/3004280`.

[8] I. Duff, P. Leleux, D. Ruiz, F. S. Torun, Row replicated block cimmino, SIAM Journal on Scientific Computing 45 (4) (2023) C207–C232. `doi:10.1137/22M1487710`.

[9] F. S. Torun, M. Manguoglu, C. Aykanat, Enhancing block cimmino for sparse linear systems with dense columns via schur complement, SIAM Journal on Scientific Computing 45 (2) (2023) C49–C72.

[10] T. Hoefler, D. Alistarh, T. Ben-Nun, N. Dryden, A. Peste, Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks, Journal of Machine Learning Research 22 (241) (2021) 1–124.

[11] W. Li, X. Song, Y. Tu, Graphdrl: Gnn-based deep reinforcement learning for interactive recommendation with sparse data, Expert Systems with Applications (2025) 126832.

[12] N. K. Unnikrishnan, J. Gould, K. K. Parhi, Scv-gnn: Sparse compressed vector-based graph neural network aggregation, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 42 (12) (2023) 4803–4816.

[13] D. Chakrabarti, Y. Zhan, C. Faloutsos, R-mat: A recursive model for graph mining, SIAM International Conference on Data MiningAccessed from foundational works on graph mining (2004).

[14] E. Abbe, Community detection and stochastic block models: Recent developments, Journal of Machine Learning Research 18 (177) (2017) 1–86.

[15] A. Farhadi, A. Taheri, Application of genai in synthetic data generation in the healthcare system, in: Application of Generative AI in Healthcare Systems, Springer, 2025, pp. 67–89.

[16] G. Huang, G. Dai, Y. Wang, H. Yang, Ge-spmm: General-purpose sparse matrix-matrix multiplication on gpus for graph neural networks, in: SC20: International Conference for High Performance Computing, Networking, Storage and Analysis, IEEE, 2020, pp. 1–12.

[17] Z. Chen, Z. Qu, Y. Quan, L. Liu, Y. Ding, Y. Xie, Dynamic n: M fine-grained structured sparse attention mechanism, in: Proceedings of the 28th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming, 2023, pp. 369–379.

[18] R. Keys, Cubic convolution interpolation for digital image processing, IEEE transactions on acoustics, speech, and signal processing 29 (6) (2003) 1153–1160.

[19] P. Thévenaz, T. Blu, M. Unser, Image interpolation and resampling, Handbook of medical imaging, processing and analysis 1 (1) (2000) 393–420.

[20] P. J. BURT, E. H. ADELSON, The laplacian pyramid as a compact image code, IEEE TRANSACTIONS ON COMMUNICATIONS 3 (4) (1983).

[21] C. E. Duchon, Lanczos filtering in one and two dimensions, Journal of Applied Meteorology (1962-1982) (1979) 1016–1022.

[22] W. L. Briggs, V. E. Henson, The DFT: an owner's manual for the discrete Fourier transform, SIAM, 1995.

[23] G. Strang, The discrete cosine transform, SIAM review 41 (1) (1999) 135–147.

APPENDIX (A)

Detail of experiments on `cavity03`. In Table 3, N denotes the number of rows and columns in the square matrix, NNZ denotes the total number of nonzero entries, and Dens is the nonzero density. Psym denotes the pattern symmetry that is the ratio of symmetric nonzero pairs. Diag denotes the number of nonzeros on the main diagonal, while Ndg counts how many diagonals contain at least one nonzero. Dist is the average distance of nonzeros from the main diagonal, normalized by matrix size. Band denotes the bandwidth, or the maximum diagonal distance of nonzero entries. Profil denotes the profile, summarizing the distance of nonzeros from the start of each row. Row-wise sparsity is represented by Rmx, Rmi, and Rstd, which are the maximum, minimum, and standard deviation of nonzeros per row, respectively. Similarly, Cmx, Cmi, and Cstd describe the same statistics for the column-wise distribution.

TABLE 3. **Details of the `cavity03` and generated matrices from it**

Operation	Method	N	NNZ	Dens	Psym	Diag	Ndg	Dist	Band	Profil	Rmx	Rmi	Rstd	Cmx	Cmi	Cstd
-	Original	317	7327	0.07	0.80	243	92	30.6	102	27652	62	8	12.5	62	1	18.1
Expand	Bilinear	318	1125	0.01	0.09	38	83	32.4	101	16286	14	0	2.5	18	0	2.9
Expand	DCT	318	6583	0.07	0.71	216	92	30.8	102	27409	62	0	13.0	61	0	15.9
Expand	DFT	318	63459	0.63	0.84	302	314	87.5	313	95691	306	4	59.4	305	1	63.1
Expand	Image	318	7384	0.07	0.81	244	92	30.8	102	27899	63	7	12.6	63	1	18.2
Expand	Lanczos	318	7781	0.08	0.44	191	100	33.4	103	36229	55	3	10.1	57	0	12.4
Expand	NN	318	7331	0.07	0.80	243	92	30.6	102	27737	62	0	12.5	62	0	18.1
Expand	Wavelet	318	6583	0.07	0.71	216	92	30.8	102	27409	62	0	13.0	61	0	15.9
Upscale	Bilinear	634	1749	0.00	0.04	29	171	63.0	205	53337	13	0	2.2	14	0	2.6
Upscale	DCT	634	118019	0.29	0.95	634	224	80.1	223	172778	304	58	56.7	304	80	60.8
Upscale	DFT	634	116754	0.29	0.75	544	634	163.8	633	294609	391	0	102.8	390	0	112.9
Upscale	Image	634	29244	0.07	0.81	486	185	61.2	205	111234	124	14	25.0	124	2	36.1
Upscale	Lanczos	634	9542	0.02	0.29	242	192	65.6	207	120037	47	1	9.0	51	0	10.7
Upscale	NN	634	13015	0.03	0.53	333	182	61.1	204	111105	75	2	13.7	80	0	18.1
Upscale	Wavelet	634	49618	0.12	0.86	520	191	67.8	206	130071	187	12	38.9	185	0	46.0
Reduce	Bilinear	316	1105	0.01	0.07	35	88	33.0	101	17078	15	0	2.6	13	0	2.9
Reduce	DCT	316	6573	0.07	0.71	215	90	30.8	101	27335	62	4	12.9	61	0	15.9
Reduce	DFT	316	75019	0.75	0.80	307	313	89.0	312	96850	303	165	40.1	314	169	38.3
Reduce	Image	316	7240	0.07	0.80	242	92	30.4	102	27403	62	7	12.4	62	1	17.9
Reduce	Lanczos	316	7811	0.08	0.45	183	99	33.1	105	35996	61	5	10.3	59	1	12.2
Reduce	NN	316	7292	0.07	0.80	240	92	30.6	102	27630	62	8	12.4	62	0	17.9
Reduce	Wavelet	316	6561	0.07	0.71	215	92	30.7	102	27314	62	4	12.9	61	0	15.8
Downscale	Bilinear	158	521	0.02	0.15	28	45	16.4	51	4154	12	0	2.3	13	0	2.8
Downscale	DCT	158	7342	0.29	0.95	158	55	20.0	54	10632	76	15	14.2	76	20	15.2
Downscale	DFT	158	23586	0.94	0.94	158	158	51.9	157	24779	157	130	4.6	158	135	4.7
Downscale	Gaussian	159	6668	0.26	0.94	159	54	19.6	53	10213	68	15	13.0	68	4	14.7
Downscale	Image	158	2662	0.11	0.85	134	47	16.2	51	7383	46	5	8.9	46	1	11.3
Downscale	Lanczos	158	3191	0.13	0.56	133	51	17.2	51	9177	38	6	7.3	41	1	8.7
Downscale	NN	158	2126	0.09	0.70	111	47	15.9	51	7295	34	2	6.6	35	0	8.9
Downscale	Wavelet	158	2708	0.11	0.85	125	43	16.4	47	7188	42	0	9.5	40	0	10.6