

# EGO: Efficient Compression of Unstructured Sparse DNNs for Compute-in-Memory based on Graph Minimum-Cost Matching Optimization

Teng Wan, Yu Cao, Huazhong Yang, Xueqing Li

Department of Electronic Engineering, LFET/BNRist/SKLSNC, Tsinghua University

Corresponding Email: xueqingli@tsinghua.edu.cn

**Abstract**—Compute-in-memory (CiM) for edge AI inference operates under strict memory and energy constraints. While unstructured pruning reduces model size and computation, efficiently deploying the sparse weights on CiM’s dense, regular arrays remains challenging. Existing studies either incur high indexing overhead by storing per-element indexing metadata, or achieve limited compression by relying on scarce structural patterns within unstructured weights. The column packing method, which avoids the high overhead of per-element indexing and offers rich compression potential, shows promise to reconcile unstructured sparsity with CiM’s regular compute pattern, but its direct application to CiM is hindered by heuristic grouping algorithms that either yield suboptimal compression or sacrifice model accuracy.

To bridge this gap and unlock the potential of column packing for CiM, this study presents EGO, an algorithm-hardware co-designed framework. EGO overcomes the inefficiency of heuristic grouping by introducing a combinatorially optimized grouping algorithm, which formulates column packing as minimum-cost graph matching. A digital CiM architecture is co-designed with the EGO column grouping formulation, which features a custom Sparsity Processing Unit (SPU) to enable efficient activation routing while preserving CiM’s dense and regular dataflow. Circuit-level simulations show that EGO achieves 1.4–3.7x average improvement in energy efficiency and 1.2–1.8x average improvement in area efficiency compared to previous state-of-the-art methods.

**Index Terms**—unstructured sparsity, compute-in-memory, pruning, artificial intelligence

## I. INTRODUCTION & BACKGROUND

Edge AI applications such as autonomous vehicles, robotics, and wearable devices demand fast, accurate, and energy-efficient neural network inference. Unstructured pruning [1]–[3] has been proven effective in reducing model size and computation while preserving accuracy, making it ideal for resource-constrained deployment. However, efficiently deploying unstructured sparse models onto CiM architectures [4]–[8] remains challenging, as CiM relies on **dense, regular data layouts** to achieve high throughput and energy efficiency, as show in Fig. 1(a).

As shown in Fig. 1(b), some methods [9] store only nonzero weights with per-element indices to compress sparse weights into a dense format, but results in high relative indexing logic and memory overhead for weight decoding or activation routing, which degrades overall energy efficiency. Some other studies [10] identify structural patterns within unstructured

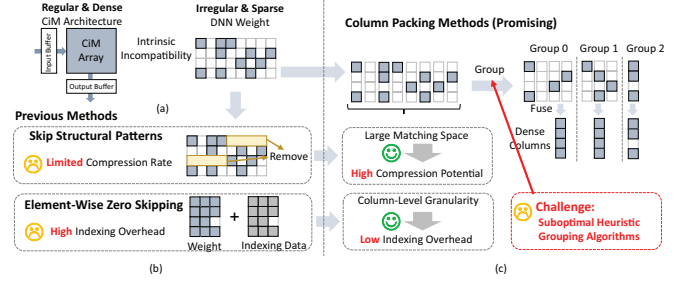


Fig. 1. (a) Unstructured sparsity preserves accuracy but is intrinsically incompatible with CiM’s regular arrays. (b) Existing methods either incur high indexing overhead or suffer from limited compression rate. (c) The column packing method avoids the high overhead of per-element indexing and offers rich compression potential, but is hindered by the suboptimal heuristic grouping algorithms.

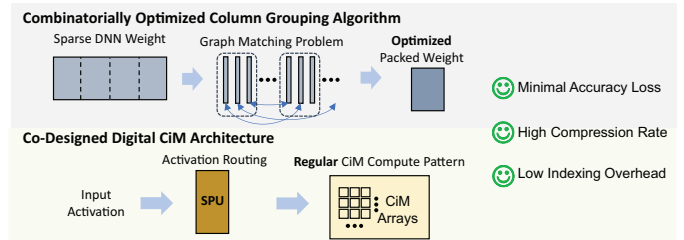


Fig. 2. This study presents EGO, a column packing framework that comprises a combinatorially optimized grouping algorithm and a co-designed CiM architecture that enables efficient, high-accuracy deployment of unstructured sparse DNNs on CiM.

pruned weights for weight compression, but the compression opportunities are limited by the inherent scarcity of such patterns.

An alternative strategy, referred to as **column packing** [11], [12], offers a more promising path to reconcile the irregularity of unstructured sparsity with the regularity required by CiM arrays, which is shown in Fig. 1(c). The column packing method groups sparse weight columns whose nonzero elements occupy mutually disjoint rows, and compresses weights by fusing each group into a single dense column. This approach operates at column-level granularity, eliminating the need for costly per-element indexing. In addition, the large combinatorial space of potential column matches within a weight matrix shows promise to achieve high compression rates.

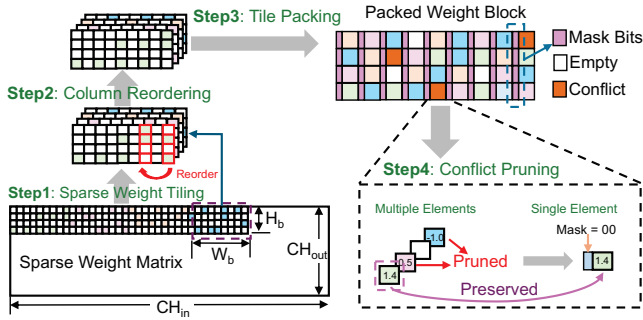


Fig. 3. Illustration of the weight permuting and packing process. The sparse weight matrix is partitioned into tiles, reordered to minimize conflicts, and fused into dense blocks. Conflicts are resolved by retaining the weight with the highest pruning score, followed by fine-tuning to recover accuracy.

However, directly applying column packing to CiM exposes a critical challenge. Prior studies rely on heuristic grouping algorithms that fail to fully explore the combinatorial search space [11], [12]. As a result, these methods either produce suboptimal groupings that yield limited compression rates, or accept approximate matches which sacrifices model accuracy.

To exploit the potential of column packing for CiM, this study presents EGO, a column packing framework, as shown in Fig. 2, which comprises:

- 1) a **combinatorially optimized** column grouping algorithm that formulates the grouping problem as a minimum-cost matching over a multipartite graph. The problem is decomposed into bipartite subproblems solvable with mathematical optimality via the Kuhn-Munkres algorithm [13].
- 2) a CiM architecture **co-designed** with the grouping formulation, featuring a Sparsity Processing Unit (SPU) that dynamically routes input activations to match pre-packed weight columns, preserving the dense, regular dataflow essential for CiM efficiency while requiring minimal modification to the core compute array.

The co-designed EGO framework is evaluated across sparsity levels from 50% to 80%, achieving up to  $1.5\text{--}1.6\times$  average compression rate compared to previous methods, while incurring less than 1% accuracy loss during the compression. Circuit-level simulations show the presented EGO achieves  $1.4\text{--}3.7\times$  average energy efficiency and  $1.2\text{--}1.8\times$  average area efficiency compared to previous state-of-the-art solutions.

## II. METHOD

### A. Graph Minimum-Cost Matching Formulation

a) *Weight Packing Overview:* As illustrated in Fig. 3, EGO partitions the sparse weight matrix into fixed-size tiles of shape  $(H_b, W_b)$ . To enable efficient fusion into dense blocks while preserving computational correctness, EGO packs only tiles that span disjoint input channels but share the same output channels. This constraint ensures that outputs from packed tiles can be directly summed to reconstruct the original

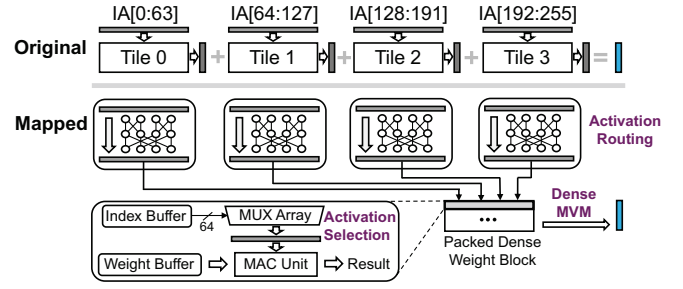


Fig. 4. Compute flow for packed weights: Input activations from multiple tiles are reordered and dynamically selected using mask bits, forming a dense vector that multiplies with the packed weight block, preserving standard MVM.

layer output, enabling fusion without altering the functional behavior of the network.

Each group of  $N$  such tiles is packed into a single dense block of shape  $(H_b, W_b)$ . To track the origin of each nonzero element, a  $\lceil \log_2 N \rceil$ -bit mask is stored per element, indicating which source tile it belongs to. The tiling strategy of EGO is different from prior work [11] which performed packing on the entire weight.

Reference [11] permuted the rows of each layer’s weight to align with the columns of the next layer, eliminating the expensive activation router for activation routing. However, it is valid only for linear layers and  $1 \times 1$  convolutions, and is therefore not used in this study.

When multiple tiles contribute nonzeros to the same position in the packed block, a conflict occurs. To resolve conflicts, EGO retains only the weight with the highest pruning score, setting others to zero (while prior work [11] used magnitude alone). The accuracy is subsequently recovered via lightweight fine-tuning on the training dataset (within 5 epochs).

b) *Compute Flow:* The packed representation preserves the native matrix-vector multiplication (MVM) structure of CiM arrays, as shown in Fig. 4. During inference:

- 1) Input activations are reordered to align with the permuted columns of each source tile.
- 2) For each position in the packed weight block, the Sparsity Processing Unit (SPU) uses the stored mask bits to dynamically select the corresponding activation from among the  $N$  candidate inputs (one per source tile).
- 3) The selected activations form a dense vector, which is multiplied with the packed dense weight block using standard MVM units.

This process ensures compatibility with existing CiM compute pipelines, requiring no modification to the core array, only requiring preprocessing of input activations.

c) *Graph-Minimum-Cost-Matching Formulation:* The core challenge in packing is to permute columns within each tile to minimize conflicts after fusion. This is formulated as a minimum-cost matching problem over a multipartite graph, as shown in Fig. 5.

For two tiles, the problem reduces to bipartite matching: each column in tile A is a node in set  $X$ , each column in tile

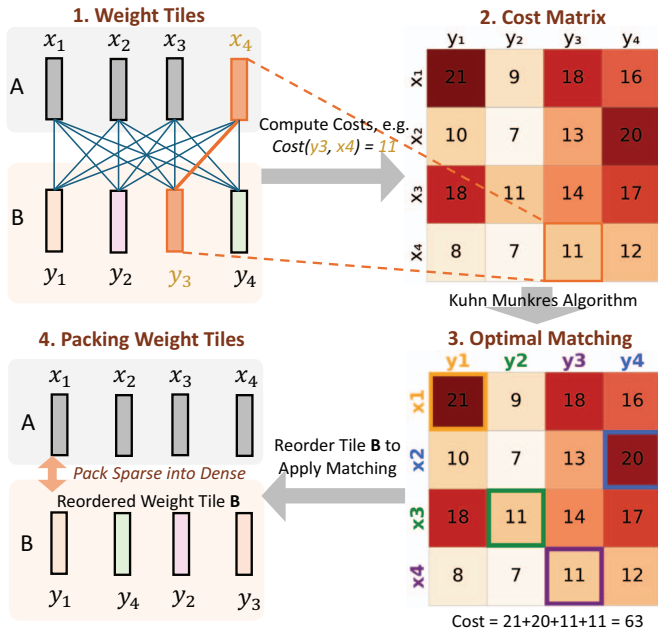


Fig. 5. Graph formulation of the column permutation problem. Packing two tiles is modeled as bipartite matching; packing  $N$  tiles is modeled as minimum-cost matching in an  $N$ -partite graph.

$B$  is a node in set  $Y$ . The edge cost between column  $i$  in  $X$  and column  $j$  in  $Y$  is defined as the squared sum of pruning scores of weights that would be lost if these columns are packed. This bipartite formulation is solved optimally using the Kuhn-Munkres algorithm [13], which provides a mathematically provable optimal matching.

To scale to  $N$  tiles, the problem is generalized to an  $N$ -partite graph, where each partition corresponds to the column set of one tile. Though the  $N$ -partite case is NP-hard for  $N > 2$ , EGO applies a hierarchical pairwise matching strategy: it recursively matches tile pairs using the bipartite formulation, then treats each matched pair as a virtual tile for subsequent rounds. This approach guarantees optimality within each pairwise step and empirically achieves near-optimal global packing across diverse sparsity levels.

### B. CiM Macro Architecture

a) *Overall Architecture*: The presented digital CiM architecture, shown in Fig. 6, extends a digital-adder-tree-based CiM array with a custom Sparsity Processing Unit (SPU) to enable efficient execution of packed sparse weights without modifying the core compute datapath.

The SPU performs the runtime reorganization of input activations: for each position in the packed weight block, it selects the appropriate activation from among  $N$  candidate inputs according to the bit masks stored alongside the weights.

This architecture ensures that the output of the SPU is always a dense activation vector compatible with standard matrix-vector multiplication units. Thus, the core CiM array operates identically for both sparse (packed) and dense models.

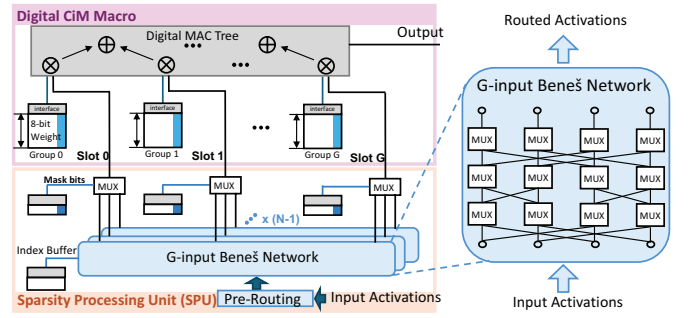


Fig. 6. The custom digital CiM architecture designed to support MVM computing with the compressed weights. The architecture contains a SPU to process input activations, which is subsequently fed to the standard digital CiM macro for MVM computation.

TABLE I  
LIST OF HYPER-PARAMETER VALUES

Parameter	Description	Value
$W_b$	Tile width	64
$H_b$	Tile height	4
$K$	Group size	8
$G$	Number of groups per block	8
$N$	Number of packed tiles per block	2,3,4,5

This feature brings opportunities for deploying models with mixed sparse and dense weights in complex systems.

b) *Activation Routing Network*: To maintain computational correctness, input activations must be reordered to match the permuted column order of the packed weights. This permutation is implemented using Beneš networks [14], as shown on the right side of Fig. 6. A  $G$ -input Beneš network enables full rearrangement of  $G$  elements within a single cycle. Each Beneš network consists of  $2\lceil \log_2 G \rceil - 1$  layers, with each layer containing  $G$  MUX units and associated SRAM-based control bits, as shown in Fig. 6.

The Beneš network's throughput is aligned with the bit-serial input consumption rate of the  $G$ -input adder tree, which maximized resource utilization. The adder tree processes one full weight per cycle, multiplying the weight with the input activation bits, which is followed by shift-and-accumulate. Each CiM block is responsible for the multiply-and-accumulate computation of a single packed weight column with the corresponding input activations,  $N - 1$  independent Beneš networks are required, one for each additional tile except the base tile, which stays at its original order.

c) *Two-Stage Column Permutation*: To reduce activation routing complexity in hardware, EGO introduces a two-stage column permutation strategy that imposes structure on the matching solution. Instead of globally permuting all  $W_b$  columns, columns are first partitioned into  $G$  groups of size  $K = W_b/G$ . Permutation is then decomposed into:

- 1) **Pre-permuting (intra-group)**: Columns within each group are permuted with the same relative order.
- 2) **Group-permuting (inter-group)**: Columns sharing the

same intra-group position (i.e., same “slot”  $k \in \{1, \dots, K\}$ ) are permuted independently across groups. This stage determines *which group contributes* to each computation slot.

Both permutation stages are optimized via the multipartite matching formulation and the Algorithm introduced in Section II-A0c, ensuring optimal column alignment within their respective constraints.

This weight permuting hierarchy further maps to a two-level activation routing scheme:

- 1) **Pre-routing** statically reorders the input activations within each group before inference, aligning them with the column order determined by pre-permuting. During execution, each group contributes exactly one activation per cycle.
- 2) **Slot-routing** dynamically selects one activation at each cycle from each group to feed its corresponding computation slot, guided by the group-permuting mask. Since selection occurs only among  $G$  candidates per slot rather than  $W_b$ , the routing logic remains lightweight.

The hyper-parameters are selected to ensure close-to-lossless accuracy and low indexing overhead, which are listed in Table. I.

### III. EVALUATION

This section evaluates the proposed framework against state-of-the-art baselines; experimental setup details are provided in Section III-A.

The evaluation focuses on two key metrics. First, the **compression rate** quantifies the reduction in required CiM array resources for DNN deployment. Second, **indexing overhead** measures the additional hardware and energy needed to support computation on the compressed representation. In cases where indexing overhead is not the system bottleneck, compression rate serves as a strong proxy for energy and area efficiency.

Section III-B presents accuracy-compression trade-offs. Section III-C breaks down indexing-related area and power costs. Section III-D then evaluates end-to-end energy and area efficiency at the macro level, incorporating both compression and overhead effects.

#### A. Experimental Setup

*a) Baselines:* Four state-of-the-art methods for accelerating unstructured sparse DNNs are selected for comparison, which include the column packing methods directly adapted from previous studies [11], [12] to CiM. These baselines, along with their abbreviations used throughout the evaluation, are summarized in Table II. The abbreviations will be used throughout this section. All methods are re-implemented under identical hardware constraints to ensure fair comparison.

*b) Workloads:* Three representative CNN architectures, which are: GoogLeNet [15], ResNet18 [16], and ResNet34 [16], are evaluated to cover diverse model sizes, redundancy levels, and layer dimensions. All models are trained on ImageNet [17], pruned using Optimal Brain Compression

TABLE II  
BASELINE CiM-COMPATIBLE ARCHITECTURES FOR COMPARISON

Baseline	Method
CC [11]	Greedy Column Packing w/ Conflict Pruning
TC [12]	SA-Optimized Column Packing w/o Conflict Pruning
BZS [9]	Element-Wise Zero Skipping
HGWR [10]	Block-Wise Zero Skipping

TABLE III  
LIST OF HARDWARE PARAMETERS

Item	Value	Item	Value
Data Bitwidth	A8W8	MAC Tree #Input	8
MAC Tree Area	$318\mu\text{m}^2$	MAC Tree #Cells	254
Buffer Area ( $N = 4$ ) <sup>a</sup>	$447\mu\text{m}^2$	Router Area ( $N = 4$ )	$89\mu\text{m}^2$

<sup>a</sup>  $N = 4$  supports most sparsity levels

(OBC) [1], and quantized to 8-bit integers for both weights and activations.

*c) Hardware Configuration:* The core compute unit is a synthesized 8-input digital adder tree targeting a 28nm CMOS process. On-chip buffers, which include weight buffers and index buffers, are generated using a commercial SRAM compiler. Input and output routing for methods requiring activation permutation (e.g., BZS [9]) are implemented using transmission gate-based Benes routers, consistent with the original BZS design. To ensure fairness of comparison, the same router implementation is reused across all compared methods, including the presented EGO. Router power is obtained via SPICE simulation, and the router area is extracted from analog layout. Key hardware parameters are listed in Table III.

To ensure architectural fairness and eliminate implementation bias, all baseline methods are re-evaluated using the same technology. In addition, the core compute units and routing hardware used the baselines are consistent with the EGO hardware.

*d) Software Configuration:* For the CC baseline [11], the trade-off hyperparameter  $\gamma$  is set to  $0.03/(1 - \text{sparsity})$ , balancing accuracy and compression. The packing width  $\alpha$  which denotes the maximum number of columns fused into one dense column is set to 4 for 67% sparsity and 8 for higher sparsity levels to avoid compression bottlenecks. Hyperparameters for HGWR [10] follow the original specification. For BZS [9], both the original 32-to-8 (25% compression) and an extended 32-to-16 configuration (50% compression) are evaluated to explore performance under relaxed compression.

#### B. Model Accuracy and Compression Rate

*a) Accuracy:* Fig. 7 compares model accuracy across the DNN workloads. The *Original* baseline denotes unstructured pruned models without compression-induced accuracy loss. EGO achieves accuracy within 1% of this baseline, outperforming CC, which used greedy column matching. HGWR, which applies block-4 semi-structured pruning, incurs up to

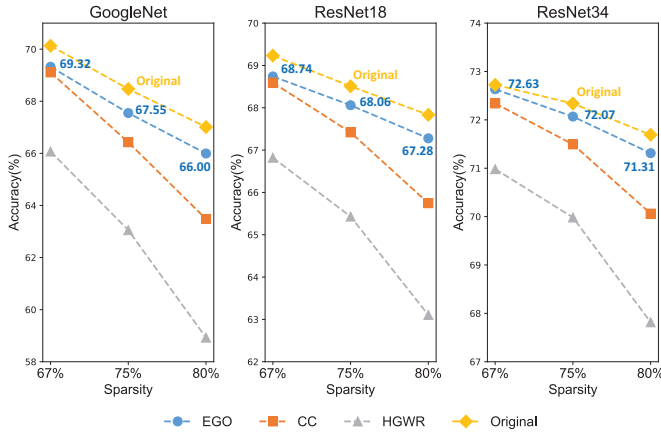


Fig. 7. Model accuracy comparison. *Original* includes lossless compression baselines **BZS** and **TC**. EGO shows within 1% accuracy while enabling hardware-efficient deployment.

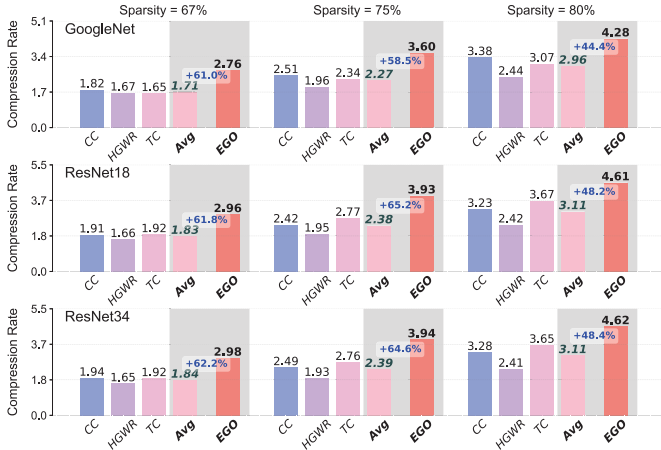


Fig. 8. Compression rate comparison. EGO achieves the highest compression across all workloads and sparsity levels, approaching the theoretical limit.

2% accuracy degradation. The accuracy advantage of EGO over CC stems directly from its combinatorially optimized column grouping, which minimizes pruning loss during conflict resolution.

*b) Compression Rate:* Fig. 8 reports the compression rate, defined as the reduction in required CiM array resources for DNN deployment. BZS is excluded from this comparison, as its compression is trivially determined by sparsity  $1/(1 - \text{sparsity})$  and does not reflect algorithmic efficiency. Later evaluations will reveal that the high compression rate of BZS comes at the price of high indexing overhead, which severely damages overall energy efficiency and area efficiency.

EGO achieves compression rates close to the theoretical upper bound across all sparsity levels, with minor deviation arising from tile boundary underutilization in the CiM array. Packing-based methods (EGO, CC and TC) consistently outperform block-wise zero-skipping (HGWR), demonstrating the efficiency of column packing.

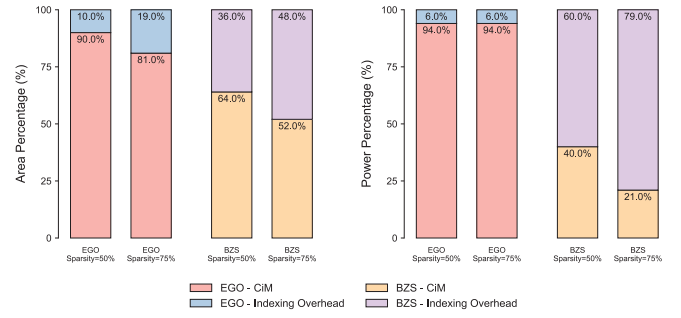


Fig. 9. Power and area breakdown. BZS [9] incurs high indexing overhead, degrading overall efficiency. EGO minimizes this overhead through router sharing and optimized packing.

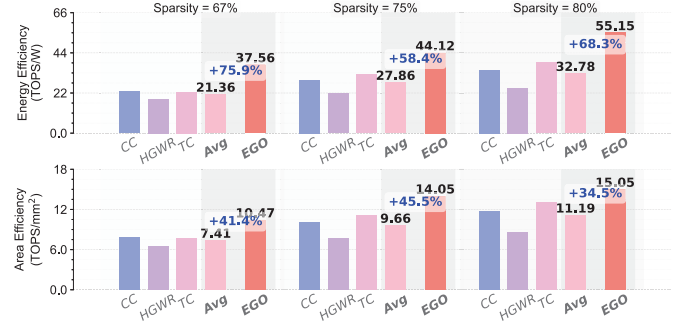


Fig. 10. Comparison of macro-level energy efficiencies and area efficiencies.

### C. Indexing Overhead Analysis

Fig. 9 compares the indexing overhead, which is defined as percentage of total macro power and area. The comparison is performed for EGO and BZS, at 50% and 75% sparsity levels respectively. In BZS, indexing dominates power consumption (60% and 79%) and occupies nearly half the macro area, which is a result of its per-adder-tree routing architecture. In contrast, EGO reduces indexing overhead significantly by sharing each router across  $H_b$  adder trees, which incurs minimal accuracy loss.

### D. Macro-Level Evaluations

Fig. 10 compares energy efficiency (TOPS/W) and area efficiency (TOPS/mm²) across sparsity levels, which reflects the overall optimizations. The workload used for the comparisons is ResNet-34. At the macro level, EGO consistently outperforms all other baselines, achieving 1.3-1.7x average improvement. The BZS baseline is not included in this figure because its sparsity levels are limited to 50% and 75%.

Table IV compares the energy efficiency, area efficiency and accuracy at a fixed 75% sparsity, which includes all baselines. EGO achieves 1.4–3.7x energy efficiency and 1.2–1.8x area efficiency over prior state-of-the-art methods. These gains result from the synergistic reduction in both weight storage and indexing logic.

TABLE IV  
ENERGY EFFICIENCY AND AREA EFFICIENCY EVALUATIONS.

Method	Energy Efficiency (TOPS/W)	Area Efficiency (TOPS/mm <sup>2</sup> )	Accuracy of ResNet34(%)
<b>EGO</b>	<b>44.1</b>	<b>14.1</b>	<b>72.1</b>
CC [11]	29.2	10.1	71.5
HGWR [10]	22.0	7.7	70.0
TC [12]	32.4	11.2	72.3
BZS [9]	11.8	9.7	72.3

#### IV. DISCUSSIONS AND FUTURE WORK

##### A. Conclusion

This study presents EGO, a column packing framework comprising a combinatorially optimized grouping algorithm and a co-designed CiM architecture that unlocks the potentiation of column packing for efficient edge DNN deployment on CiM architectures. EGO achieve 1.4–3.7x improvement in effective energy efficiency and 1.2–1.8x improvement in effective area efficiency compared to previous state-of-the-arts.

The gains of EGO stem from two synergistic innovations: (1) a graph-based, combinatorially optimized formulation of column packing that minimizes accuracy loss, and (2) a lightweight Sparsity Processing Unit (SPU) that enables standard matrix-vector multiplication on packed weights, preserving the dense CiM computing dataflow.

##### B. Future Work

While EGO’s pairwise matching strategy is effective and grounded in provable optimization, future work may explore multi-partite matching algorithms to further reduce accuracy loss during compression.

Additionally, extending EGO to attention-based architectures, such as Vision Transformers and integrating it with dynamic sparsity patterns such as sparse attention and activation sparsity represent promising directions to broaden its applicability beyond static CNN workloads.

#### V. ACKNOWLEDGMENT

This work is supported in part by NSFC(Grant #U24B6015, #92264204, and #21B2030), and in part by National Key R&D Program of China (Grant #2024YFB3214500)

#### REFERENCES

[1] E. Frantar, S. P. Singh, and D. Alistarh, “Optimal brain compression: a framework for accurate post-training quantization and pruning,” in *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS ’22, (Red Hook, NY, USA), Curran Associates Inc., 2024.

[2] E. Frantar and D. Alistarh, “Sparsegpt: massive language models can be accurately pruned in one-shot,” in *Proceedings of the 40th International Conference on Machine Learning*, ICML’23, JMLR.org, 2023.

[3] H. Cheng, M. Zhang, and J. Q. Shi, “A survey on deep neural network pruning: Taxonomy, comparison, analysis, and recommendations,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 46, no. 12, pp. 10558–10578, 2024.

[4] J.-W. Su, Y.-C. Chou, R. Liu, T.-W. Liu, P.-J. Lu, P.-C. Wu, Y.-L. Chung, L.-Y. Hung, J.-S. Ren, T. Pan, S.-H. Li, S.-C. Chang, S.-S. Sheu, W.-C. Lo, C.-I. Wu, X. Si, C.-C. Lo, R.-S. Liu, C.-C. Hsieh, K.-T. Tang, and M.-F. Chang, “16.3 A 28nm 384kb 6T-SRAM Computation-in-Memory Macro with 8b Precision for AI Edge Chips,” in *2021 IEEE International Solid-State Circuits Conference (ISSCC)*, vol. 64, pp. 250–252, Feb. 2021. ISSN: 2376-8606.

[5] X. Si, Y.-N. Tu, W.-H. Huang, J.-W. Su, P.-J. Lu, J.-H. Wang, T.-W. Liu, S.-Y. Wu, R. Liu, Y.-C. Chou, Z. Zhang, S.-H. Sie, W.-C. Wei, Y.-C. Lo, T.-H. Wen, T.-H. Hsu, Y.-K. Chen, W. Shih, C.-C. Lo, R.-S. Liu, C.-C. Hsieh, K.-T. Tang, N.-C. Lien, W.-C. Shih, Y. He, Q. Li, and M.-F. Chang, “15.5 A 28nm 64Kb 6T SRAM Computing-in-Memory Macro with 8b MAC Operation for AI Edge Chips,” in *2020 IEEE International Solid-State Circuits Conference - (ISSCC)*, pp. 246–248, Feb. 2020. ISSN: 2376-8606.

[6] J. Wang, X. Wang, C. Eckert, A. Subramaniyan, R. Das, D. Blaauw, and D. Sylvester, “14.2 A Compute SRAM with Bit-Serial Integer/Floating-Point Operations for Programmable In-Memory Vector Acceleration,” in *2019 IEEE International Solid-State Circuits Conference - (ISSCC)*, pp. 224–226, Feb. 2019. ISSN: 2376-8606.

[7] Y.-D. Chih, P.-H. Lee, H. Fujiwara, Y.-C. Shih, C.-F. Lee, R. Naoas, Y.-L. Chen, C.-P. Lo, C.-H. Lu, H. Mori, W.-C. Zhao, D. Sun, M. E. Sinangil, Y.-H. Chen, T.-L. Chou, K. Akarvardar, H.-J. Liao, Y. Wang, M.-F. Chang, and T.-Y. J. Chang, “16.4 An 89TOPS/W and 16.3TOPS/mm<sup>2</sup> All-Digital SRAM-Based Full-Precision Compute-In Memory Macro in 22nm for Machine-Learning Edge Applications,” in *2021 IEEE International Solid-State Circuits Conference (ISSCC)*, vol. 64, pp. 252–254, Feb. 2021. ISSN: 2376-8606.

[8] H. Mori, W.-C. Zhao, C.-E. Lee, C.-F. Lee, Y.-H. Hsu, C.-K. Chuang, T. Hashizume, H.-C. Tung, Y.-Y. Liu, S.-R. Wu, K. Akarvardar, T.-L. Chou, H. Fujiwara, Y. Wang, Y.-D. Chih, Y.-H. Chen, H.-J. Liao, and T.-Y. J. Chang, “A 4nm 6163-TOPS/W/b \mathbf{4790-TOPS/mm}^2/b SRAM Based Digital-Computing-in-Memory Macro Supporting Bit-Width Flexibility and Simultaneous MAC and Weight Update,” in *2023 IEEE International Solid-State Circuits Conference (ISSCC)*, pp. 132–134, Feb. 2023. ISSN: 2376-8606.

[9] S. Liu, P. Li, J. Zhang, Y. Wang, H. Zhu, W. Jiang, S. Tang, C. Chen, Q. Liu, and M. Liu, “16.2 a 28nm 53.8tops/w 8b sparse transformer accelerator with in-memory butterfly zero skipper for unstructured-pruned nn and cim-based local-attention-reusable engine,” in *2023 IEEE International Solid-State Circuits Conference (ISSCC)*, pp. 250–252, 2023.

[10] L. Wu, C. Zhao, J. Wang, X. Yu, S. Chen, C. Li, J. Han, X. Xue, and X. Zeng, “A heuristic and greedy weight remapping scheme with hardware optimization for irregular sparse neural networks implemented on cim accelerator in edge ai applications,” in *Proceedings of the 29th Asia and South Pacific Design Automation Conference, ASPDAC ’24*, p. 551–556, IEEE Press, 2024.

[11] H. Kung, B. McDanel, and S. Q. Zhang, “Packing sparse convolutional neural networks for efficient systolic array implementations: Column combining under joint optimization,” in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 821–834, 2019.

[12] X. Chen, J. Zhu, J. Jiang, and C.-Y. Tsui, “Tight compression: Compressing cnn model tightly through unstructured pruning and simulated annealing based permutation,” in *2020 57th ACM/IEEE Design Automation Conference (DAC)*, pp. 1–6, 2020.

[13] F. Serratos, “Fast computation of bipartite graph matching,” *Pattern Recognition Letters*, vol. 45, pp. 244–250, 2014.

[14] P. D. Manuel, M. I. Abd-El-Barr, I. Rajasingh, and B. Rajan, “An efficient representation of benes networks and its applications,” *Journal of Discrete Algorithms*, vol. 6, no. 1, pp. 11–19, 2008. Selected papers from AWOCA 2005.

[15] C. Szegegy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, “Going deeper with convolutions,” in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1–9, 2015.

[16] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[17] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, “Imagenet: A large-scale hierarchical image database,” in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 248–255, 2009.