

# GMaC: NvCIM Architecture for Parallel Point-based Point Cloud Acceleration via Geometric Mapping and Address-Index Computation

Yi Gao<sup>1</sup>, Zongwei Wang<sup>1,2\*</sup>, Ling Liang<sup>1</sup>, Yimao Cai<sup>1</sup>

<sup>1</sup>School of Integrated Circuits, Beijing Advanced Innovation Center for Integrated Circuits, Peking University, Beijing, China

<sup>2</sup>YanXin MicroElectronics Co., Ltd. (YXME), Shanghai, China \*Corresponding author: wangzongwei@pku.edu.cn

**Abstract**—3D point cloud is essential for spatial understanding and 3D modeling in autonomous driving, robotics, and AR/VR. Due to the efficient processing of unordered point clouds, Point-based Point Cloud Neural Networks (PNNs) have attracted significant attention, with accelerators targeting two primary stages: down-sampling and feature computation. However, the rapid growth of point cloud data exposes severe memory-wall bottlenecks in traditional von Neumann architectures and challenges in supporting intensive MVM operations. Existing CIM architectures also lack efficient support for Euclidean distance computation, incurring additional data movement and programming overhead. To address the above challenges, we propose GMaC, an nvCIM architecture for parallel PNN acceleration. GMaC introduces a hardware-friendly geometric mapping algorithm to enable parallel execution between down-sampling and feature computation. A novel RRAM-based address-index computation circuit is further proposed to accelerate Euclidean distance calculations in the analog domain for down-sampling. We validate GMaC with a hardware simulator based on the RRAM-CIM platform. Experimental results demonstrate that GMaC achieves  $3.57\times$  speedup and  $4.96\times$  energy savings compared to the state-of-the-art ASIC designs, with negligible accuracy loss. These findings highlight the potential of nvCIM technology in edge-based PNN implementation.

**Index Terms**—computing-in-memory, non-volatile memory, point-based point cloud acceleration, parallel architecture

## I. INTRODUCTION

Point clouds represent 3D objects as unordered point sets, enabling realistic scene simulations and finding wide applications in spatial exploration, autonomous driving, robotics, and AR/VR. With the cost of point collection devices (e.g., LiDAR) decreasing, point cloud technology shows great promise for edge deployment, including portable devices like assistive sunglasses for the visually impaired [1]. However, these advances require higher system efficiency and lower latency for real-time processing.

Point-based neural networks (PNNs), such as PointNet, have become a key research focus due to their efficient handling of unordered points and simple architectures [2], [3]. To address the non-uniform distribution of points, PointNet++ [4] introduces geometric extractors like farthest point sampling (FPS) and query ball grouping, further enhancing learning through multi-layer feature extraction. Recent studies mainly optimize two stages in PNNs: Euclidean distance-based down-sampling and MVM-based feature computation (Fig. 1(a)).

As point cloud data (PCD) scales up, von Neumann-based edge processors face increasing memory wall constraints [5].

This work was supported by the NSFC (62322401, U24B20166, 62341407, 62406008) and “111” Project (B18001).

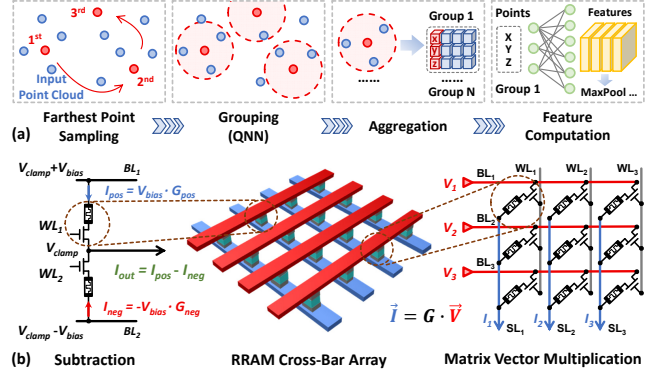


Fig. 1. (a) Sequential processing architecture of Pointnet++; (b) Implementation of MVM and subtraction operations based on 1T1R RRAM array.

Irregular memory access of PCD and frequent movement of large MLP weights result in significant latency and energy overhead. Conventional CMOS designs struggle to meet the rising demand for MVM operations, which dominate feature computation and are highly hardware-dependent. Upgrading to high-performance hardware (e.g., GPUs) or integrating additional MAC units introduces substantial power, area, and data transfer costs. Both computational and memory overheads grow sharply with PCD size (Fig. 2(a)). Recent works have introduced nv-CIM technologies, especially RRAM-based CIM [5]–[8], to efficiently accelerate MVM within crossbar arrays for PNNs (Fig. 1(b)), alleviating the memory wall.

Despite these advances, significant challenges persist. **① At the algorithm level, current point cloud processing methods—particularly sophisticated down-sampling extractors—are hardware-unfriendly for edge deployment.** While dedicated local extractors improve PNN inference robustness, they often incur over 50% latency overhead [9]–[12] due to reliance on high-precision, large-scale PCD. This accuracy-latency trade-off is especially problematic in resource-constrained scenarios, with diminishing accuracy gains from further iterations. Approximate search strategies can significantly improve system performance [13]–[15] (Fig. 2(b)), but current approaches lack comprehensive system-level algorithm-hardware co-optimization, limiting achievable network accuracy. **② At the circuit level, the absence of efficient Euclidean distance operators offsets the benefits of CIM architectures in addressing the memory wall.** As shown in Fig. 2(c), the acceleration of MVM shifts the system bottleneck to Euclidean distance calculations during down-sampling, resulting in non-uniform system-level optimization.

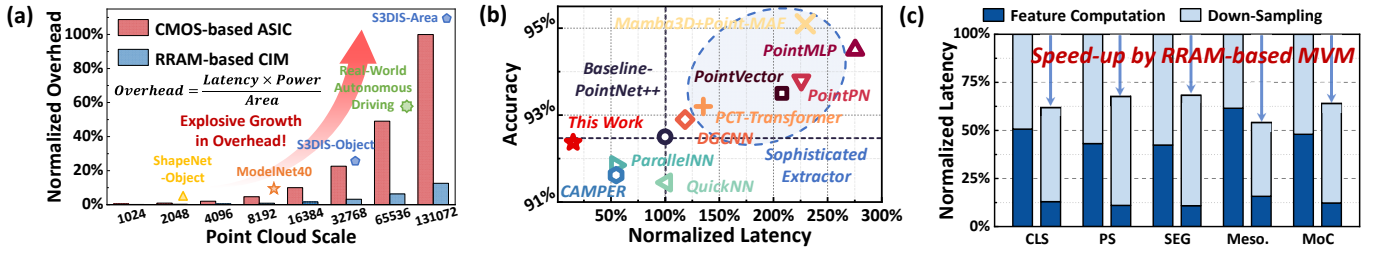


Fig. 2. (a) Existing works exhibit a trade-off between latency and accuracy; (b) RRAM-based CIM demonstrates advantages in large-scale MVM computations; (c) Normalized time consumption proportion of down-sampling and feature computation on GPU and ASICs with efficient MVM acceleration through nvCIM.

Existing works (e.g., PointCIM) offer preliminary explorations but do not address RRAM programming overhead. Despite its significant impact on overall performance, optimization using emerging nvCIM has received limited attention. **At the dataflow level, most PNN accelerators adopt a sequential structure, causing performance bottlenecks to accumulate during execution.** Limited MVM throughput leads to strong data dependency between down-sampling and feature computation, and large input batches are required for efficient GPU/TPU utilization, enforcing sequential execution of both stages. Such designs lead to prohibitive latency for customized edge systems and stack bottlenecks. Notably, down-sampling and feature computation target different aspects of PCD—global spatial information and local geometric features, respectively—suggesting potential for parallel architecture.

In this work, we propose GMaC, an RRAM-CIM-based parallel architecture for efficient PNN acceleration with algorithm-circuit-architecture co-design. Our main contributions are as follows:

- **Algorithm:** We propose the Geometric Mapping (GM) algorithm, a hardware-friendly optimization tailored for edge point cloud processing. Through a lightweight mapping operation, GM shifts the FPS search from the raw PCD to fixed neighboring points, reducing computational complexity with minimal accuracy loss.
- **Circuit:** Based on the RRAM-CIM platform, we propose specialized circuits for GM, along with a novel address-index computation circuit for efficient analog-domain Euclidean distance calculation. Combined with high-throughput MVM acceleration, this approach delivers holistic system-level optimization for CIM architectures.
- **Dataflow:** We achieve parallel down-sampling and feature computation by PCD decomposition via GM. This separation, together with a scalable double-pointer buffer for global aggregation, enables parallel processing and mitigates nearly half of the performance bottlenecks.
- **System:** We implement an ASIC design based on RRAM-CIM to support GMaC. Comprehensive benchmarks show GMaC achieves  $3.57\times$  speedup and  $4.96\times$  energy savings over SOTA ASICs, demonstrating strong potential for nvCIM-based point cloud acceleration.

## II. BACKGROUND

### A. Point-based Neural Network

Point-based neural networks process 3D point clouds by directly extracting features from input points. PointNet++

serves as a foundational framework, comprising the following key steps: **Farthest Point Sampling (FPS):** FPS selects the  $m$  farthest points from  $n$  samples, effectively reducing sample size while preserving global features, outperforming random sampling in retaining global information. **Grouping:** After FPS, neighboring points are selected via KNN [16] or radius search around the  $m$  sampled points, forming groups for further processing. FPS and grouping together achieve down-sampling. **Aggregation:** Points within each group are mapped to relative coordinates, enabling efficient local feature aggregation. **Feature Computation:** Aggregated points are processed by a shared multi-layer perceptron (MLP) to extract local features, which are then combined into global representations via pooling (e.g., max-pooling).

### B. PNN Accelerator

Recent efforts to accelerate PNN have targeted specific processing stages. PointX [17] and PRADA [18] leverage clustering to extract spatial features and simplify local feature computation, respectively. Mesorasi [19] reduces feature computation costs via delayed aggregation, shifting bottlenecks to aggregation. EdgePC [20] employs Morton encoding to skip redundant computations in structured point clouds. PointAcc [21] implements FPS and KNN using a generic ranking-based kernel. MARS [22] introduces FPS-stage filtering to minimize off-chip data access. However, most approaches optimize isolated steps, leading to bottleneck accumulation due to sequential execution. Moreover, sophisticated extractor designs often reduce PNN’s adaptability and scalability.

### C. RRAM-based Computing in Memory

Recent advances in non-volatile memory (NVM) have enabled efficient edge computing for neural networks, with RRAM-based CIM technology showing great potential [23]–[27]. RRAM’s non-volatility allows in-situ computation and weight storage, eliminating the need for data transfer and minimizing memory access in PNNs. Its low-voltage operation, high density, and multilevel capability significantly reduce hardware and energy costs, making it well-suited for edge PNN deployment. The crossbar structure enables efficient analog matrix-vector multiplication, alleviating bottlenecks in aggregation and feature computation.

## III. ALGORITHM

### A. Revisiting PNN Method

Point cloud data consists of 3D coordinates, with each point represented as  $(x, y, z)$ . For a set of points  $p_i$ , PointNet++

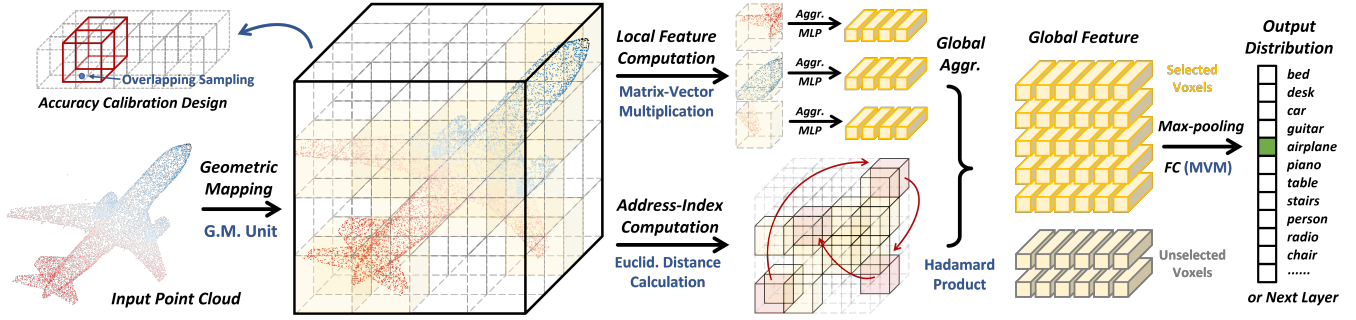


Fig. 3. An overview of GMaC Algorithm. The GM algorithm decouples raw points into global and local components, which are processed in parallel by the down-sampling and feature computation units. Euclidean distance computation based on fixed voxel positions significantly reduces hardware overhead.

vides a classical point-based framework that has demonstrated outstanding performance across tasks such as classification and segmentation. Its workflow is abstracted as:

$$p_j = \mathcal{D}(p_i) \quad i = 1, 2, \dots, n \quad (1)$$

$$f_j = \mathcal{F}(\mathcal{A}(\mathcal{N}(p_j), p_j)) \quad j = 1, 2, \dots, m \quad (2)$$

where  $\mathcal{D}$  denotes down-sampling via FPS, selecting  $m$  maximally distant points from input  $n$  points. Neighbor search  $\mathcal{N}$  groups points around each sampled  $p_j$ , followed by aggregation  $\mathcal{A}$  and feature extraction  $\mathcal{F}$  to obtain point cloud features  $f_j$ . For classification, fully connected layers and activations yield the final probability distribution. Down-sampling ( $\mathcal{D}$  &  $\mathcal{N}$ ) and feature computation ( $\mathcal{F}$ ) dominate latency and energy, with sequential execution intensifying bottlenecks.

### B. Proposed Geometric Mapping Algorithm

Based on PointNet++, we propose GMaC (Fig. 3), introducing a geometric mapping step  $\mathcal{M}$  that partitions the input point cloud into predefined local regions (e.g., spheres or voxels). This decouples each point into relative local coordinates  $\mathbf{p}_i$  and absolute region locations  $V_k$ , represented as:

$$\mathcal{M}(p_i) = \mathbf{p}_i \odot V_k \quad k = 1, 2, \dots, l \quad (3)$$

Since  $\mathcal{D}$  captures global spatial relationships and  $V_k$  replaces the absolute position of  $p_i$ , applying  $\mathcal{D}$  to local  $\mathbf{p}_i$  is redundant. Substituting  $\mathcal{M}(p_i)$  into Eq.(1) yields:

$$\mathcal{D}(\mathcal{M}(p_i)) = \mathbf{p}_i \odot \mathcal{D}(V_k) \quad (4)$$

Similarly,  $\mathcal{N}$ ,  $\mathcal{A}$ , and  $\mathcal{F}$  operate on local coordinates, making further processing on  $V_k$  unnecessary. Substituting  $\mathcal{D}(\mathcal{M}(p_i))$  into feature computation yields:

$$f_i = \mathcal{F}(\mathcal{A}(\mathcal{N}(\mathbf{p}_i), \mathbf{p}_i)) \odot \mathcal{D}(V_k) \quad (5)$$

Through  $\mathcal{M}$ , points in each region  $V_k$  are grouped for aggregation. Adjusting region size and introducing overlapping sampling ensures sufficient points per region, approximating the  $\mathcal{N}$  operation:

$$f_i \approx \mathcal{F}(\mathcal{A}(\mathbf{p}_i | \mathbf{p}_i \in V_k)) \odot \mathcal{D}(V_k) \quad (6)$$

The lightweight  $\mathcal{M}$  step benefits multiple PNN stages, as the Hadamard product integrates local and global information to provide a richer representation, decoupling feature computation dependency from down-sampling.

**Parallel Workflow.** Geometric mapping separates local coordinates and global locations, enabling parallel MLP and

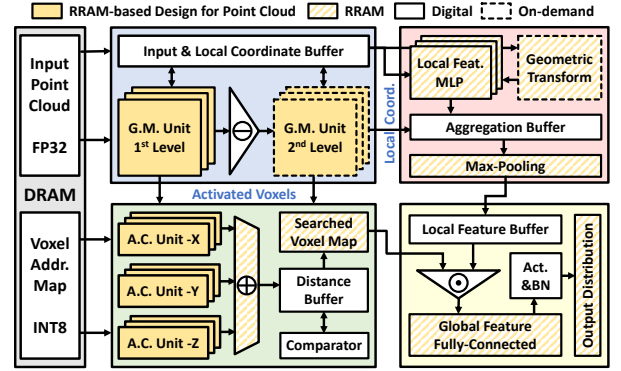


Fig. 4. An Overview of GMaC Architecture.

FPS computation in dedicated hardware units, thus eliminating sequential bottlenecks.

**Hardware Friendly.** Decoupling PCD reduces the data volume of each part, allowing low-precision types without accuracy loss. Instead of raw FP32 PCD, FPS operates on region locations, typically divided into no more than 16 segments per dimension, enabling efficient INT8 computation and lowering hardware cost. Based on reduced spatial scale, local feature MLPs support FP16 to enhance energy efficiency.

**Memory Access.** FPS operates exclusively on the relative locations of activated regions, thereby eliminating dependencies on the raw point cloud and avoiding redundant memory accesses and reducing buffer requirements.

**Adaptability.** Modifying region types in  $\mathcal{M}$  and shifting extractors to  $V_k$  allows GMaC to adapt to various point cloud optimizations flexibly. For example, the GM with spherical region division for PointNet++ is defined as:

$$\sqrt{(x_i - x_k)^2 + (y_i - y_k)^2 + (z_i - z_k)^2} \leq R \quad (7)$$

where  $(x_k, y_k, z_k)$  is the center and  $R$  the radius. This maintains accuracy while shifting computation workload to the pre-design phase, requiring no extra on-chip resources.

**Flexibility.** Different  $\mathcal{M}$  schemes strike a balance between accuracy, efficiency, and speed. To highlight GMaC's advantages, we propose a voxel-based GM with RRAM-CIM-based circuit design, dividing input space into voxels and calibrating accuracy via overlapping sampling, as detailed next.

## IV. ARCHITECTURE

As shown in Fig.4, the proposed GMaC point cloud processing architecture consists of the following key units: geometric

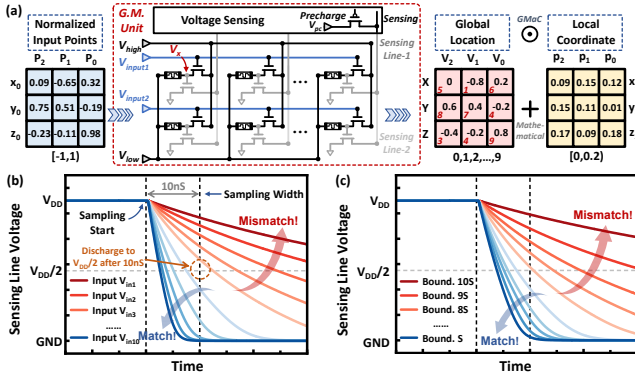


Fig. 5. (a) Circuit design for geometric mapping based on 2T1R RRAM array and (b)&(c) simulation results of geometric mapping based on discharge rate.

mapping, voxel address-index computation, and local & global feature computation. Based on the algorithm in Section 3, we detailed the implementation of each unit and simulation results, with comprehensive system evaluations in Section V.

### A. Geometric Mapping

As shown in Fig. 5(a), we implement geometric mapping using a 2T1R RRAM array, where each row partitions the  $x$ ,  $y$ , or  $z$  axes into  $n$  segments ( $n \leq 16$ ), forming  $n^3$  voxels of length  $S$ . In each 2T1R cell, the 1T1R unit (color black) stores voxel boundaries. Applying a bias voltage to both ends yields a different intermediate voltage  $V_x$  for varying  $V_{in}$ , controlling discharge rates on the sensing line through the grey transistor. A threshold of  $\frac{1}{2}V_{DD}$  with a 10ns response time is set for boundary detection. As shown in Fig. 5(b), when the fifth boundary  $5S$  is mapped into the RRAM, inputs above  $5S$  (blue lines) discharge below  $\frac{1}{2}V_{DD}$  within 10ns, while those below  $5S$  (red lines) do not. For each row, consecutive voxel boundaries are mapped into RRAM cells. Applying the same  $V_{in}$  across columns yields different  $V_x$  divisions, enabling input range identification (e.g.,  $[5S, 6S]$  in Fig. 5(c) with first 5 matched devices). The segmentation number  $n$  is easily adjusted by adjusting RRAM weights, allowing flexible resolution. Parallel operations along  $Y$  and  $Z$  complete the mapping for each point.

After geometric mapping, normalized voxel coordinates  $V_k = (X, Y, Z)$  are activated in the Searched Voxel Map and Distance Buffer for address-index computation. Subtracting voxel locations from input points yields local coordinates for feature extraction. Unlike Voxel CNNs, which replace point features with voxels, our method uses voxels only for spatial structure, preserving point information.

### B. Address-Index Computation for FPS

Voxel-based farthest sampling utilizes distance table calculations and lookups, enabling hardware support for diverse point cloud algorithms. After geometric mapping,  $m$  voxels are activated in the Searched Voxel Map, defined by integer relative locations  $(X, Y, Z)$ . Euclidean distance computation involves numerous subtractions and multiplications, leading to significant hardware overhead in digital circuits and programming overhead in existing RRAM-based solutions. Based

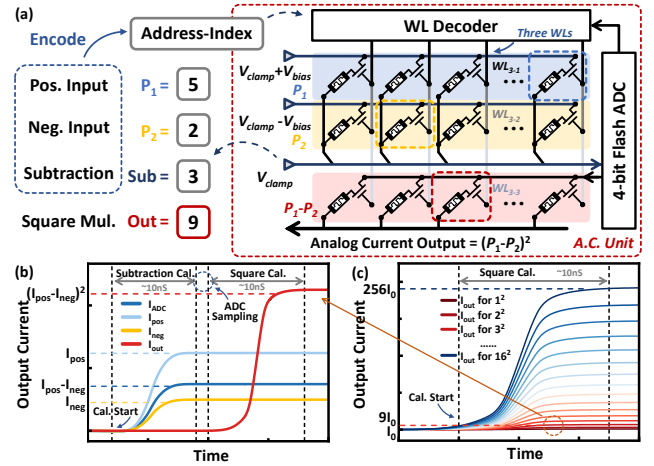


Fig. 6. (a) Address-index computation circuit for Euclidean distance calculation based on 1T1R RRAM array. (b)&(c) Simulation output current.

on the conversion from floating-point to integer-based FPS via GM, we propose the RRAM-based address-index computation. As shown in Fig. 6(a), a three-row 1T1R RRAM array is configured, with columns  $L_n$  matching the number of voxels  $n$ . RRAM devices are programmed to specific conductance states, storing weights corresponding to physical column addresses:

$$W_n = \alpha \times L_n \quad (8)$$

For any two voxel coordinates  $X_1$  and  $X_2$ , the positive (first row) input word line (WL) at column  $X_1$  and the negative (second row) input WL at column  $X_2$  are activated. Through differential inputs and voltage clamping on the source line (SL), the analog calculation current  $X_1 - X_2$  is obtained by Kirchhoff's law. This SL current is then quantized by a shared Flash ADC and fed into the bit-line (BL) of the third row. Simultaneously, the weight device in the column  $|X_1 - X_2|$  is activated to perform the multiplication of BL input and RRAM conductance:

$$I_{out} = V_{BL} \times G_{RRAM} = (X_1 - X_2)^2 \quad (9)$$

Currents from all three dimensions are accumulated, completing Euclidean distance computation for FPS comparison. This approach matches physical column addresses with stored data, enabling efficient data retrieval via address-index encoding. The three-terminal 1T1R structure supports in-situ CIM computation, with WL for data extraction and BL for voltage application. Data type optimization and accelerated computation reduce down-sampling overhead, providing a robust foundation for efficient ASIC dataflow.

### C. Accuracy Calibration Design

**High Resolution.** To improve accuracy, GMaC employs two-stage sampling in geometric mapping. The first stage uses coarse-grained mapping steps  $S_1$  to generate local coordinates, followed by fine-grained mapping with step  $S_2$  ( $S_2 < S_1$ ), dividing the space into  $(S_1 \times S_2)^3$  voxels for high-resolution and large-scale point clouds with hierarchical configuration.

**Overlapping Sampling.** Points located on voxel boundaries are split into separate regions, hindering effective learning. We introduce overlapping voxels to address boundary effects by reducing the second-stage step to  $\frac{1}{2}S_1$ , forming new voxels

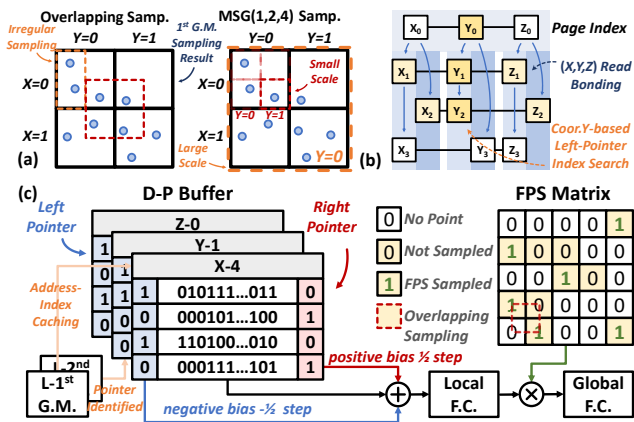


Fig. 7. (a) Schematic of multi-scale grouping during aggregation or overlapping sampling. (b) Index search with Y-based left pointer. (c) Efficient global aggregation can be achieved by D-P buffer and bias settings with FPS result.

from adjacent halves. Each point obtains two local coordinate representations, aggregated into different groups. This allows each point to reside in two overlapping voxels per dimension, mitigating boundary-induced feature splitting and enhancing feature consistency and group-level information to eliminate the negative influence of random voxel splitting on feature extraction accuracy.

#### D. Scalable Double-Pointer Buffer

In PNNs such as PointNet++ with multi-scale grouping (MSG) or overlapping sampling, the same point cloud often requires multiple representations centered at different points (Fig.7(a)) and independent aggregation, increasing on-chip storage overhead. To address this, we propose a scalable double-pointer buffer that reconstructs multi-scale data from single-scale coordinates to reduce storage requirements.

As shown in Fig. 7(b), buffer pages are index-constructed by relative voxel coordinates after the first-stage GM, with X, Y, and Z stored separately but linked by shared positions. The second-stage GM assigns left/right pointers: the left pointer covers  $[0, \frac{1}{2}S_1]$ , and the right pointer covers  $[\frac{1}{2}S_1, S_1]$ . During buffer output, data can be flexibly indexed along any dimension using these pointers. For example, to query the left half-region along the X-Z plane, buffer spaces where the Y page’s left pointer is set are sequentially accessed, retrieving X and Z data via positional binding. Fig. 7(c) shows how left/right pointers introduce biases during aggregation, enabling multi-scale feature reconstruction and global aggregation with FPS results. This partitions 3D space into eight subspaces, supporting efficient re-aggregation via neighboring regions. The approach achieves low hardware cost and optimal balance of computational complexity and resource use.

### V. EVALUATION

#### A. Evaluation Setup

**Baseline.** We evaluate GMaC using PointNet++ on classification (cls), part segmentation (ps), and semantic segmentation (seg) tasks with ModelNet40 [28], ShapeNet [29], and S3DIS [30], respectively. ModelNet40 (10,000 points/object) was

TABLE I  
GMaC HARDWARE IMPLEMENTATION

Components	Scale	Frequency	Area(mm <sup>2</sup> )
G.M. Unit	3×16(Array)	100MHz	0.01
A.C. Unit	3×16(Array)×4	100MHz	0.01
4bit Flash ADC	1×4	1GHz	0.04
Comparator	512	1GHz	0.018
RRAM macro	12Mb	100MHz	7.8
SRAM Buffer	256KB	/	0.62
<b>Sum Area</b>	<b>15.7 (PointAcc)</b>	<b>7 (MoC)</b>	<b>8.5</b>

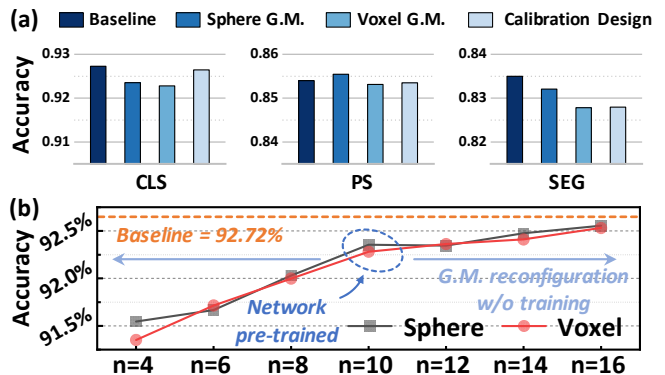


Fig. 8. (a) The accuracy of GMaC algorithm in different tasks. (b) The influence of divisions  $n$  on network accuracy.

specially chosen to simulate realistic LiDAR scan volumes, highlighting GMaC’s efficiency for large-scale point clouds. To fully emphasize the advantages of NvCIM architecture, GMaC is compared against CPU (Intel Xeon Platinum 8360Y), GPU (Nvidia RTX 4090), SOTA CMOS-based ASICs (Mesorasi [19], PointAcc [21], MARS [22], MoC [31]), and CIM-based accelerators (Voxel-CIM [32], PointCIM [5]).

**GMaC Implementation.** For fair comparison, GMaC maintains a consistent hardware area across all implementations (see Table I). A 28nm 12Mb RRAM macro supports 16-bit floating-point computation, as referenced in SOTA designs [24], with an on-chip global buffer. For each 10,000-point frame, serialized RRAM-based MLPs allow in-place data overwrite, requiring only 128KB for maximum feature output. Additional buffers include 2KB for FPS distance iterations and storage for input points and local coordinates. Analog units were simulated using the Stanford-PKU RRAM model [33] in Virtuoso (28nm node), and digital modules were synthesized with Synopsys Design Compiler (28nm CMOS). Data access efficiency was evaluated using DRAMsim3 [34] for off-chip simulation and CACTI [35] for normalized on-chip buffering.

#### B. Accuracy

To assess GMaC’s impact on inference accuracy, we implemented the modified algorithm in PyTorch [36] and evaluated three tasks on GPUs. As shown in Fig. 8(a), with spherical GM and  $n = 10$  divisions along XYZ axes, the retrained model accuracy closely matches the baseline. Using voxels introduces minor accuracy loss, but performance remains acceptable and still strong for part and semantic segmentation. Further improvements are achieved by increasing voxel divisions and applying overlapping sampling. On average, GMaC

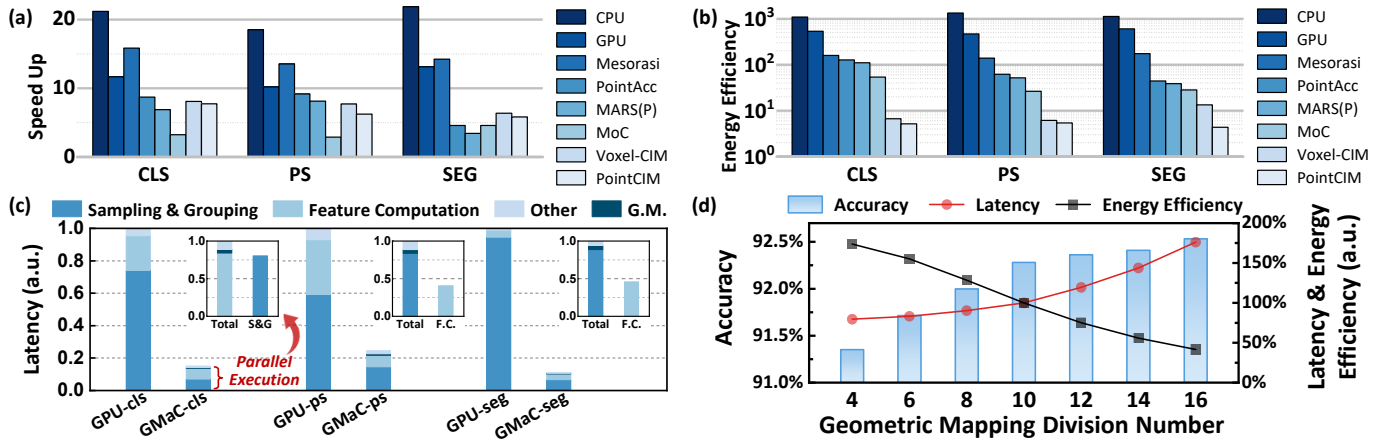


Fig. 9. (a) The speedup and (b) energy savings of GMaC over CPU, GPU, and ASICs, respectively. (c) Simulation results of latency breakdown. (d) Design space exploration: the influence of G.M. division  $n$  on system accuracy, latency, and energy efficiency.

incurs only 0.58% (sphere) and 0.63% (voxel) accuracy loss (Fig. 8(b)), confirming suitability for PointNet++ and derived networks. GMaC has the potential to serve as a fundamental architecture for developing accurate point cloud algorithms. For subsequent evaluations,  $n = 10$  is used, with accuracy loss versus baseline reduced to 0.44%.

### C. Speedup Performance

Fig. 9(a) shows GMaC achieves average speedups of  $20.53\times$ ,  $11.66\times$ ,  $19.29\times$ ,  $7.49\times$ ,  $6.15\times$ ,  $3.57\times$ ,  $7.38\times$ , and  $6.60\times$  over CPU, GPU, Mesorasi, PointAcc, MARS(P), MoC, Voxel-CIM, and PointCIM, respectively. Latency breakdown (Fig. 9(c)) indicates GMaC reduces local computational complexity and exploits parallelism, unlike traditional designs focusing on individual components. In GMaC, system latency is mainly determined by feature computation, as down-sampling is executed in parallel, mitigating nearly half of the bottleneck—an improvement difficult for current accelerators that optimize only one stage. Hardware resource allocation further balances latency across components.

During down-sampling, coarse-grained analog address-index computation significantly reduces latency. In feature computation, CIM’s parallelism compensates for frequency limitations of RRAM-CIM. According to SOTA research [24], further performance gains are expected, highlighting the strong potential of RRAM-based CIM for point cloud acceleration.

### D. Energy Efficiency

Compared to CPU, GPU, and ASICs, GMaC achieves average energy savings of  $1178\times$ ,  $535\times$ ,  $157\times$ ,  $77.8\times$ ,  $66.9\times$ ,  $36.2\times$ ,  $8.72\times$ , and  $4.96\times$ , respectively (Fig. 9(b)), highlighting the efficiency of RRAM-based CIM. The crossbar structure

and non-volatile characteristic of RRAM enable simultaneous weight retrieval and in-situ matrix-vector multiplication, with partial results directly accumulated on the source line.

The novel address-index computation leverages analog processing to efficiently accelerate Euclidean distance calculation, significantly reducing power consumption. GMaC requires only a single traversal of the raw point cloud, with all MLP weights stored on-chip in RRAM, reducing DRAM and SRAM access by over 99% (Fig. 10(b)). The scalable D-P Buffer also minimizes on-chip buffer requirements for multi-scale processing (Fig. 10(a)).

### E. Design Space Exploration

GMaC flexibly balances speed, energy efficiency, and accuracy by adjusting the geometric mapping division number  $n$  (Fig. 9(d)). As  $n$  increases from 4 to 16, system accuracy improves but aggregation and FPS computations become bottlenecks. To strike a balance,  $n = 10$  is recommended for real-time processing with optimal accuracy and efficiency. Multiple modes are supported and easily configured by GM units. A smaller  $n$  boosts speed for scenarios like night driving with extra reaction time, while a larger  $n$  enhances accuracy and precise decisions for complex urban environments.

## VI. CONCLUSION

In summary, we demonstrate an efficient point cloud acceleration framework based on the RRAM-based computing-in-memory platform through algorithm-circuit-architecture co-optimization. The hardware-friendly geometric mapping algorithm and dedicated circuits in GMaC support parallel execution of down-sampling and feature computation, eliminating bottleneck accumulation under sequential designs. A novel RRAM-based address-index computation circuit is implemented to accelerate Euclidean distance calculations, further enhancing computational efficiency and algorithm capability. GMaC achieves substantial speedup and energy savings over CPUs, GPUs, and state-of-the-art ASICs, demonstrating strong potential for high-performance, energy-efficient point cloud acceleration in edge AI applications.

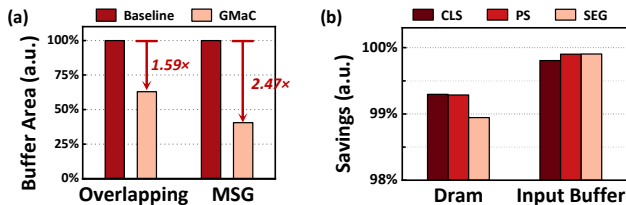


Fig. 10. (a) Normalized buffer area reduced by D-P buffer; (b) DRAM and SRAM reduction by CIM.

## REFERENCES

- [1] H. Liu, R. Liu, K. Yang, J. Zhang, K. Peng, and R. Stiefelwagen, "Hida: Towards holistic indoor understanding for the visually impaired via semantic instance segmentation with a wearable solid-state lidar sensor," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2021, pp. 1780–1790.
- [2] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "Pointnet: Deep learning on point sets for 3d classification and segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.
- [3] A. V. Phan, M. L. Nguyen, Y. L. H. Nguyen, and L. T. Bui, "Dgcnn: A convolutional neural network over large-scale labeled graphs," *Neural Networks*, vol. 108, pp. 533–543, 2018. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0893608018302636>
- [4] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "Pointnet++: Deep hierarchical feature learning on point sets in a metric space," *arXiv preprint arXiv:1706.02413*, 2017.
- [5] X.-J. Chen, H.-P. Chen, and C.-L. Yang, "Pointcim: A computing-in-memory architecture for accelerating deep point cloud analytics," in *2024 57th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2024, pp. 1309–1322.
- [6] Z. Wang and Y. Cai, "Memory technology: Development, fundamentals, and future trends," 2023.
- [7] Y. Ling, Z. Wang, Y. Yang, L. Bao, S. Bao, Q. Wang, Y. Cai, and R. Huang, "An isolated symmetrical 2t2r cell enabling high precision and high density for rram-based in-memory computing," *Science China Information Sciences*, vol. 67, no. 5, p. 152402, 2024.
- [8] Y. Ling, Z. Wang, L. Wu, Y. Cai, and R. Huang, "An rram-based hierarchical computing-in-memory architecture with synchronous parallelism for 3d point cloud recognition," *IEEE Transactions on Circuits and Systems II: Express Briefs*, 2024.
- [9] R. Zhang, L. Wang, Z. Guo, Y. Wang, P. Gao, H. Li, and J. Shi, "Parameter is not all you need: Starting from non-parametric networks for 3d point cloud analysis," *arXiv preprint arXiv:2303.08134*, 2023.
- [10] X. Deng, W. Zhang, Q. Ding, and X. Zhang, "Pointvector: A vector representation in point cloud analysis," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2023, pp. 9455–9465.
- [11] M.-H. Guo, J.-X. Cai, Z.-N. Liu, T.-J. Mu, R. R. Martin, and S.-M. Hu, "Pct: Point cloud transformer," *Computational visual media*, vol. 7, pp. 187–199, 2021.
- [12] X. Han, Y. Tang, Z. Wang, and X. Li, "Mamba3d: Enhancing local features for 3d point cloud analysis via state space model," in *Proceedings of the 32nd ACM International Conference on Multimedia*, 2024, pp. 4995–5004.
- [13] R. Pinkham, S. Zeng, and Z. Zhang, "Quicknn: Memory and performance optimization of kd tree based nearest neighbor search for 3d point clouds," in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2020, pp. 180–192.
- [14] F. Chen, R. Ying, J. Xue, F. Wen, and P. Liu, "Parallelnn: A parallel octree-based nearest neighbor search accelerator for 3d point clouds," in *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2023, pp. 403–414.
- [15] J. Zheng, L. Wu, Y. Su, J. Wang, Z. Huang, C. Chen, and Q. Liu, "Camper: Exploring the potential of content addressable memory for 3d point cloud efficient range search," in *Proceedings of the 61st ACM/IEEE Design Automation Conference*, 2024, pp. 1–6.
- [16] L. E. Peterson, "K-nearest neighbor," *Scholarpedia*, vol. 4, no. 2, p. 1883, 2009.
- [17] J.-F. Zhang and Z. Zhang, "Point-x: A spatial-locality-aware architecture for energy-efficient graph-based point-cloud deep learning," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 1078–1090. [Online]. Available: <https://doi.org/10.1145/3466752.3480081>
- [18] Z. Song, H. Lu, G. Li, L. Jiang, N. Jing, and X. Liang, "Prada: Point cloud recognition acceleration via dynamic approximation," in *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2023, pp. 1–6.
- [19] Y. Feng, B. Tian, T. Xu, P. Whatmough, and Y. Zhu, "Mesorasi: Architecture support for point cloud analytics via delayed-aggregation," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2020, pp. 1037–1050.
- [20] Z. Ying, S. Bhuyan, Y. Kang, Y. Zhang, M. T. Kandemir, and C. R. Das, "Edgepc: Efficient deep learning analytics for point clouds on edge devices," in *Proceedings of the 50th Annual International Symposium on Computer Architecture*, ser. ISCA '23. New York, NY, USA: Association for Computing Machinery, 2023. [Online]. Available: <https://doi.org/10.1145/3579371.3589113>
- [21] Y. Lin, Z. Zhang, H. Tang, H. Wang, and S. Han, "Pointacc: Efficient point cloud accelerator," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 449–461. [Online]. Available: <https://doi.org/10.1145/3466752.3480084>
- [22] X. Yang, T. Fu, G. Dai, S. Zeng, K. Zhong, K. Hong, and Y. Wang, "An efficient accelerator for point-based and voxel-based point cloud neural networks," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*, 2023, pp. 1–6.
- [23] L. Shan, L. Wu, Z. Wang, R. Xie, C. Ban, G. Yang, Q. Wang, Y. Li, H. Ma, L. Bao *et al.*, "Monolithically 3d integrated memristive bayesian neural network for intelligent motion planning," in *2024 IEEE International Electron Devices Meeting (IEDM)*. IEEE, 2024, pp. 1–4.
- [24] T.-H. Wen, H.-H. Hsu, W.-S. Khwa, W.-H. Huang, Z.-E. Ke, Y.-H. Chin, H.-J. Wen, Y.-C. Chang, W.-T. Hsu, C.-C. Lo, R.-S. Liu, C.-C. Hsieh, K.-T. Tang, S.-H. Teng, C.-C. Chou, Y.-D. Chih, T.-Y. J. Chang, and M.-F. Chang, "34.8 a 22nm 16mb floating-point rram compute-in-memory macro with 31.2tflops/w for ai edge devices," in *2024 IEEE International Solid-State Circuits Conference (ISSCC)*, vol. 67, 2024, pp. 580–582.
- [25] Q. Liu, B. Gao, P. Yao, D. Wu, J. Chen, Y. Pang, W. Zhang, Y. Liao, C.-X. Xue, W.-H. Chen, J. Tang, Y. Wang, M.-F. Chang, H. Qian, and H. Wu, "33.2 a fully integrated analog rram based 78.4tops/w compute-in-memory chip with fully parallel mac computing," in *2020 IEEE International Solid-State Circuits Conference - (ISSCC)*, 2020, pp. 500–502.
- [26] Y. Gao, Z. Wang, L. Bao, H. Zhang, J. Sun, R. Xie, L. Liang, and Y. Cai, "A hybrid monolithic 3d integration of 2t0c dram and rram chip for high-precision in-memory point cloud acceleration with ultra-fine-grained dataflow," in *2025 IEEE Symposium on VLSI Technology and Circuits (VLSI Technology and Circuits)*. IEEE, 2025, pp. 1–2.
- [27] Y. Gao, Z. Wang, R. Xie, Y. Yang, L. Bao, L. Liang, Z. Zhou, and Y. Cai, "First demonstration of in-mvm xor-decryption based on 40nm rram chip for secure point cloud processing," in *2025 International Electron Devices Meeting (IEDM)*. IEEE, 2025, pp. 1–4.
- [28] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao, "3d shapenets: A deep representation for volumetric shapes," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1912–1920.
- [29] A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su *et al.*, "Shapenet: An information-rich 3d model repository," *arXiv preprint arXiv:1512.03012*, 2015.
- [30] I. Armeni, O. Sener, A. R. Zamir, H. Jiang, I. Brilakis, M. Fischer, and S. Savarese, "3d semantic parsing of large-scale indoor spaces," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.
- [31] X. Liu, Z. Song, H. Chen, X. Li, and X. Liang, "Moc: A morton-code-based fine-grained quantization for accelerating point cloud neural networks," in *Proceedings of the 61st ACM/IEEE Design Automation Conference*, 2024, pp. 1–6.
- [32] X. Lin, S. Huang, and H. Jiang, "Voxel-cim: An efficient compute-in-memory accelerator for voxel-based point cloud neural networks," *arXiv preprint arXiv:2409.19077*, 2024.
- [33] H. Li, Z. Jiang, P. Huang, Y. Wu, H.-Y. Chen, B. Gao, X. Y. Liu, J. F. Kang, and H.-S. P. Wong, "Variation-aware, reliability-emphasized design and optimization of rram using spice model," in *2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2015, pp. 1425–1430.
- [34] S. Li, Z. Yang, D. Reddy, A. Srivastava, and B. Jacob, "Dramsim3: A cycle-accurate, thermal-capable dram simulator," *IEEE Computer Architecture Letters*, vol. 19, no. 2, pp. 106–109, 2020.
- [35] R. Balasubramonian, A. B. Kahng, N. Muralimanohar, A. Shafiee, and V. Srinivas, "Cacti 7: New tools for interconnect exploration in innovative off-chip memories," *ACM Trans. Archit. Code Optim.*, vol. 14, no. 2, Jun. 2017. [Online]. Available: <https://doi.org/10.1145/3085572>
- [36] X. Yan, C. Zheng, Z. Li, S. Wang, and S. Cui, "Pointasnl: Robust point clouds processing using nonlocal neural networks with adaptive sampling," 2020.