PIM-SUM: Fast and Reliable In-Memory Summation for Recommendation Systems

Fan Li, Ruizhi Zhu, Huize Li, Di Wu, Xin Xin*
College of Engineering and Computer Science, University of Central Florida
Orlando, FL, USA
{fan.li, ruizhi.zhu, huize.li, di.wu, xin.xin}@ucf.edu

2025 IEEE 43rd International Conference on Computer Design (ICCD) | 979-8-3315-0346-8/25/$31.00 ©2025 IEEE | DOI: 10.1109/ICCD65941.2025.00042

***Abstract*—Embedding aggregation in large-scale recommendation systems creates severe memory bandwidth bottlenecks, as each query sums many high-dimensional vectors. Bitwise-operation-based PIM can exploit subarray bandwidth, but traditional designs struggle with summation because long carry propagation limits parallelism.**

We propose PIM-SUM, an in-DRAM summation primitive for Sparse Length Sum (SLS) in recommendation workloads. PIM-SUM reformulates vector summation as column-wise accumulation via popcount, truncating carry propagation and avoiding redundant in-DRAM computation. This enables high-throughput summation using native DRAM bitwise primitives. PIM-SUM integrates a Reed-Solomon-inspired error correction at the DRAM row level. Its linearity supports parity propagation, enabling integrity checking and multi-bit error correction with low overhead. On DLRM workloads, PIM-SUM achieves up to 5.14× speedup, logarithmic I/O reduction, and large energy savings. It also reduces silent data corruption by 1778× and improves detection by over 170×. These results show PIM-SUM is a scalable, fault-tolerant, and energy-efficient solution for memory-bound inference at data center scale.

***Index Terms*—Processing-in-Memory, Recommendation Systems, Embedding Aggregation**

I. Introduction

Embedding-based recommendation models dominate large-scale e-commerce platforms [1]. Large embedding tables are used to map high-cardinality categorical inputs into dense vectors, with sizes reaching hundreds of gigabytes. Each inference retrieves multiple vectors across feature fields and aggregates them into compact slot-wise representations. In deployments, embedding aggregation causes heavy system overhead. Many background tasks—such as user vector regeneration [2] or offline candidate scoring [3]—trigger large-scale Sparse Length Sum (SLS) aggregation. These operations are memory-intensive but computationally light [1], creating bandwidth pressure and latency overhead.

Processing-in-Memory (PIM) architectures alleviate this by co-locating compute with memory and reducing data transfer [4]. DRAM-based PIM leverages row activation primitives to support bulk bitwise operations with low hardware cost. Systems like Ambit [5] and SIMDRAM [6] exploit analog DRAM subarray behaviors to execute basic logic operations. These designs are cost-effective but operate only at bit level, lacking carry support and unsuitable for sum calculations. The analog-based operation further raises reliability challenges, as more errors may occur during the in-memory computation.

To bridge the gap, designs like Pinatubo [7], Compute Cache [8], and ELP2IM [9] extend bitwise logic and reduce disruption using different memory technologies, but they remain inefficient for arithmetic. Arithmetic requires full-width numbers with carry propagation, conflicting with DRAM's row-parallel structure. Bit-serial methods incur high latency, as 64-bit addition needs 64 sequential steps, and DRAM's internal frequency is low. Column-wise storage could align with bitwise logic but hurts embedding access efficiency, causing severe bank conflicts. Thus, PIM is effective for bulk bit-parallel tasks but constrained for digit-level summation.

In this work, we revisit in-DRAM computation as a foundation for accelerating memory-bound operations such as embedding aggregation. Instead of performing full-width additions via recursive carry propagation, we propose a popcount-based reduction scheme that reformulates the operation as a column-wise counting problem. By performing in-place bitwise accumulation within each column, it 'compresses' multiple embedding vectors while avoiding immediate carry resolution—a major bottleneck in conventional arithmetic. Meanwhile, we develop a summation-compatible ECC scheme to protect against both storage- and computation-induced errors. The scheme is an extension of traditional Reed-Solomon codes and is capable of detecting and correcting multiple errors within each embedding vector.

II. Background

A. PIM Solutions and Operations

1) PIM limitations: Over the past few years, a wide range of PIM designs have emerged across various memory substrates, including DRAM [5], [9], NVM [7], and SRAM [10]. These designs generally fall into two categories: (1) The first often exploits the analog feature of the memory storage media and implements in-place computation directly in memory cells. (2) The second approach often integrates extra logic units near memory arrays to provide versatile computations [11]. The trade space between the two approaches lies in their computational capability and hardware complexity, where in-place PIM solutions are often more compatible with existing memory architecture, but they can only implement fixed, limited operations. More importantly, supporting common arithmetic operations such as addition and multiplication requires

*Corresponding author.

2576-6996/25/$31.00 ©2025 IEEE
DOI 10.1109/ICCD65941.2025.00042

numerous sequential logic cycles, which can completely offset the parallel computing benefit. Therefore, **we advocate that the utilization of bitwise PIM should be guided primarily by its potential to reduce data movement across the limited channel bandwidth, rather than by its computational capability alone**. In other words, a PIM operation should be employed only if it can effectively reduce data transfer.

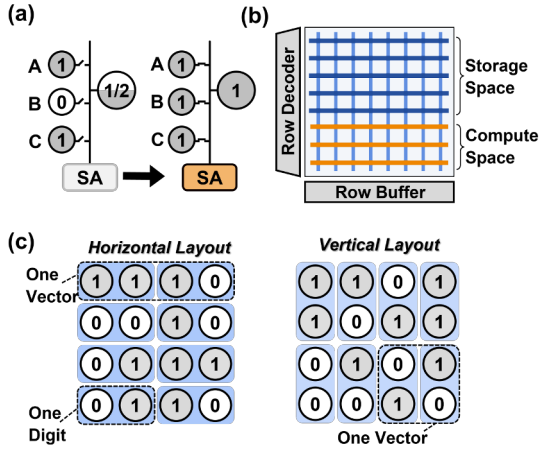

Fig. 1. DRAM-based PIM architecture (Ambit [5]): (a) the majority operation, (b) array structure, and (c) data layout

2) PIM key operations: We use Ambit as a representative example in this paper. Ambit is developed upon the RowClone technique [12], which enables row-sized data copies within the same subarray. It strategically activates a new row during the restore phase, during which the sense amplifiers have latched the cell data and drive the charges back. Thus, the latched data will overwrite the content of the newly activated row. Ambit further extends the operation of the sensing phase by simultaneously activating three rows, termed Triple Row Activation (TRA). The sensing result corresponds to the majority value of the three cells in each column. By predefining the value of one row, the other two rows can be used to perform logic operations (Fig. 1(a)). Since the TRA result will overwrite the original data, as shown in Fig. 1(b), bitwise PIM designs often divide a subarray into two separate regions for data storage and computation, respectively.

B. Embedding Aggregation in Recommendation Systems

Modern recommendation systems are a core component of large-scale online platforms such as e-commerce, video streaming, and social media. These models aim to predict user preferences based on dense numerical inputs and high-cardinality categorical features. The latter—such as user IDs, item IDs, and contextual attributes—are typically handled via large embedding tables that map discrete indices to fixed-dimensional continuous vectors.

Each inference request involves retrieving multiple embedding vectors across different feature fields, followed by a sum or mean aggregation step that compresses these vectors into a single representation per slot. The resulting vectors are concatenated and passed to downstream multilayer perceptrons (MLPs) for final prediction. In industrial-scale models, a single query may access tens to hundreds of embeddings, with each slot typically containing 20∼100 indices.

III. Motivation

While bitwise PIM solutions have the potential to accelerate embedding aggregation, they often face significant challenges in terms of computational efficiency and bandwidth utilization.

1) Horizontal Data Layout: In commodity memory, embedding vectors are stored row-wise (Fig. 1(c)) to maximize row buffer hits. A straightforward PIM strategy is bit-serial addition, processing one bit at a time from LSB to MSB. This keeps data format compatibility but wastes bandwidth, since only one bit per digit is processed per step. It also needs extra in-array logic for shifting and carry propagation, making latency scale linearly with operand width.

2) Vertical Data Layout: The vertical layout stores all bits of an element in a column. This avoids shifting but forces each vector to span multiple rows, increasing access latency. More critically, vertical storage even reduces row-level parallelism. For instance, a 4 Kb row can store two 2 Kb vectors in the horizontal layout (embedding vectors of 1–4 Kb are often comparable to DRAM row size [13]). Now, with the vertical layout and assuming each element is 16 bits, a 4 Kb row would hold 32 vectors (each storing only one bit per element). Note that no parallelism can be exploited among these 32 vectors, since recommendation systems typically access only a limited number of vectors in a nearly random manner. As shown in Fig. 1(c), the horizontal layout allows any two of the four vectors to be aggregated. In contrast, the vertical layout works only when the selected vectors are vertically aligned.

Summary: Both layouts fail to deliver efficiency; we aim to overcome this while preserving the default horizontal layout.

IV. PIM-SUM Design

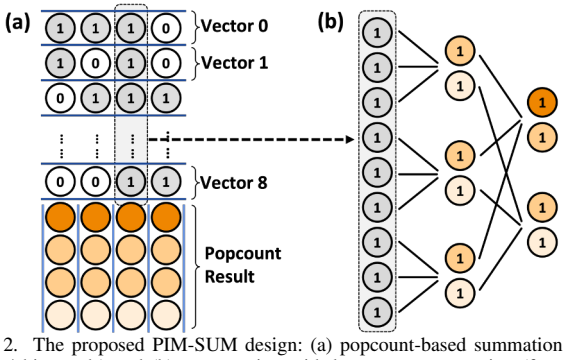

Fig. 2. The proposed PIM-SUM design: (a) popcount-based summation (of nine 4-bit words) and (b) popcounting with less carry propagation (for nine 1s)

As aforementioned, we propose PIM-SUM as an auxiliary accelerator to offload bottleneck aggregation operations in recommendation systems. It is designed to alleviate data movement overhead rather than performing all computations within memory. Subsequent operations are delegated to conventional processors such as CPUs or GPUs.

A. Efficient In-Array Summation

Drawing on the understanding of existing bitwise PIM techniques, we propose a more compatible mechanism that performs summation via distinct popcount operations, maximizing bitwise operation parallelism. In contrast to conventional PIM solutions that aim to provide final summation results, our approach instead computes only intermediate results and avoids unnecessary operations that do not contribute to reducing memory readout.

1) Performing Popcount-Based Summation: Given N embedding vectors stored across N rows, each bit column can be processed independently. In particular, by counting the number of ones (a.k.a., Popcounting) at each bit position across all N rows, we obtain an exact column-wise partial sum. These counts fall within the range $[0, N]$ and can be represented with only $\lceil \log_2 N \rceil$ bits. This means the aggregation of N vectors requires only $\lceil log_2 N \rceil$ rows per bit column. For example, aggregating 9 rows of digits requires only 4 rows to store the counting results, with each result stored column-wise, as shown in Fig. 2(a).

2) Reduction-Oriented Popcount: Another key optimization is to avoid unnecessary carry propagation during the popcounting process. To count N bits, the longest carry propagation chain can be $\lceil \log_2 N \rceil$. Whereas, many of the propagation operations do not facilitate the reduction of output data. For example, consider counting 8 bits. Counting the first 7 bits produces a result that fits within 3 bits. Including the 8th bit increases the count to a maximum of 8, which requires 4 bits to represent the result and introduces 3 carry propagations. However, this final step is redundant and only adds unnecessary complexity, as outputting the 3-bit result (for the first 7 bits) along with the separate 8th bit yields the same total output bit width.

Recall that our objective is to reduce the amount of data transfer rather than computing the final result. To this end, we truncate carry propagation by restructuring the counting operation into a ternary reduction tree. As shown in Fig. 2(b), assuming a column of popcounting bits consists entirely of 1s, the reduction process forms a two-stage tree, where each node is a full adder that compresses three input bits into two output bits. In the first stage, 9 bits are reduced to 6. In the second stage, the lower 3 bits (LSBs) and upper 3 bits (MSBs) of this 6-bit output are each further compressed into two bits, resulting in a 4-bit final output.

Only this 4-bit result is read out, and the final count (`0b'1101`) can be reconstructed by processors in a pipelined manner (by calculating $(\texttt{0b'11}) \times 2 + \texttt{0b'11}$). This tree-structured reduction further minimizes carry propagation within each popcounting operation, ensuring that each computation effectively reduces a specific amount of data transfer.

B. Computation-Compatible ECC Scheme

To protect in-DRAM vector aggregation, we incorporate a lightweight RS-inspired error correction scheme that generalizes traditional binary Reed–Solomon (RS) codes to operate over integers (a prime field $\mathbb{F}_q$). This design is compatible with the popcount-based summation in PIM-SUM and introduces minimal disruption to the memory layout or controller logic.

At the core of our ECC scheme is a parity-generation matrix $G \in \mathbb{F}_q^{r \times k}$, where k denotes the number of elements in each embedding vector, r represents the number of parity symbols, and q is a prime number chosen to match the bit-width of each element. Each row of G defines a linear combination over the k elements, producing one parity symbol. The parity symbols corresponding to an embedding vector $\vec{v} \in \mathbb{F}_q^k$ is then given by $\vec{p} = G \cdot \vec{v}$, leading to redundancy overhead of r/k. For example, a row in G like $[1, 2, 3]$ encodes a parity value as $a_0 + 2a_1 + 3a_2$. When G is chosen as a full-rank Vandermonde, the resulting code achieves minimum distance $d = r + 1$. This enables the detection of up to r errors or correction of up to $\lfloor \frac{r}{2} \rfloor$ errors.

The key feature of the proposed code is its homomorphic property: the parity of the aggregated result equals the sum of the original parities. Particularly, given two embedding vectors $\vec{a}, \vec{b}$, and their encoded parities $\vec{p}_a = G \cdot \vec{a}$, $\vec{p}_b = G \cdot \vec{b}$, the aggregated vector $\vec{c} = \vec{a} + \vec{b}$ has parity $\vec{p}_c = \vec{p}_a + \vec{p}_b$. This entails that parity information can propagate naturally through in-DRAM summation without extra computation.

In PIM-SUM, embedding vectors and parity are stored in the same row in DRAM. During aggregation, results are accumulated within the compute space, and corresponding parity accumulations are performed in parallel. Verification is carried out during the result access. Conceptually, the verification process is to first recompute the codeword parity as $\vec{p}_r = G \cdot \vec{c}$, and then compare it with the received $\vec{p}_c$ which was computed in-DRAM. The difference forms the syndrome vector $\vec{s} = \vec{p}_{c\_verif} - \vec{p}_c = G \cdot \vec{e}$, where $\vec{e}$ is the unknown error vector. Error correction thus reduces to solving a linear system $A \cdot \vec{e} = \vec{s}$, where A is derived from G.

V. Evaluation

A. Methodology

1) Performance: We use Facebook DLRM [14] as the benchmark, with SLS slot size set to 80. Batch sizes vary from 512 to 4096 to model cold-start and steady-state phases. Datasets include Anime [15], MovieLens [16], LastFM [17], and three Amazon subsets [18]. Baselines are RecNMP, RecPIM-Opt–8, and SIMDRAM (horizontal and vertical layouts).

We modify Ramulator2 [19] to simulate baseline DRAM and PIM designs, driven by Intel Pin traces. Configurations follow DDR5 JEDEC specifications [13]. To improve locality, we adopt software LRU-based page tracking and replicate hot subarrays across banks, reducing contention and boosting throughput.

2) Reliability: We compare three designs: Traditional ECC-PIM (SECDED), Non-ECC PIM, and PIM-SUM with Reed–Solomon-based row-level ECC. Errors are categorized as Correctable Errors, DUE (Detectable but Uncorrectable Errors), and SDC (Silent Data Corruption, arising from miscorrection or uncorrectable errors). Soft errors are injected as random bit flips at row level with rates from 10^{-6} to 10^{-4}.

Outputs are validated against fault-free references to measure detection and correction effectiveness.

B. Results

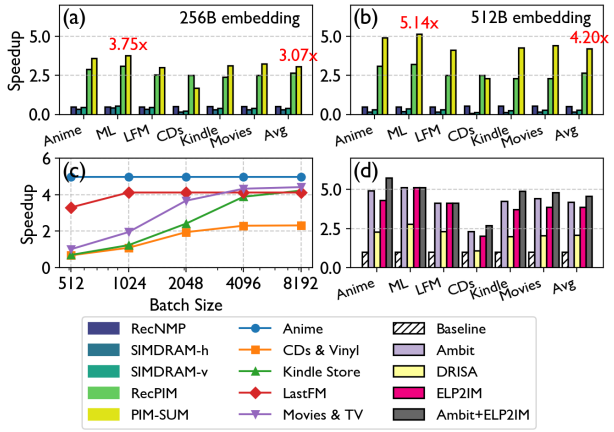

Fig. 3. PIM-SUM performance evaluation: (a–b) SLS performance across workloads, (c) scalability with batch size, and (d) SLS performance across different PIM substrates.

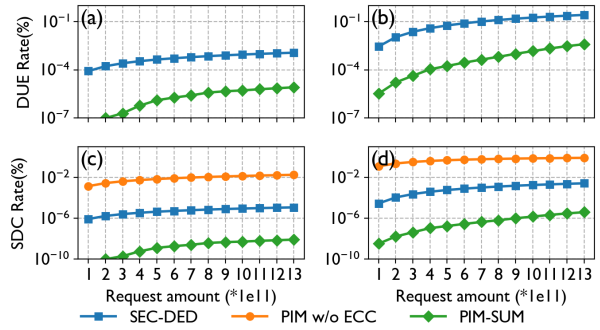

Fig. 4. Reliability evaluation under transient faults: (a)(c) fault rates at bit-flip rate 10^{-6}, (b)(d) fault rates at bit-flip rate 10^{-8}.

1) Performance: Fig. 3(a,b) shows that PIM-SUM outperforms baselines across embedding sizes, with 2.5× speedup at 128B, up to 5.14× at 512B, and 4.2× on average. Larger embeddings better match DRAM row size, improving utilization and parallelism. SIMDRAM-h is limited by bit-serial execution, SIMDRAM-v by row fragmentation, and RecPIM by irregular accesses. Fig. 3(c) shows scalability with batch size, with gains plateauing beyond 2048. Fig. 3(d) demonstrates portability, with average speedups of 4.6× on Ambit+ELP2IM, 4.2× on Ambit, 3.9× on ELP2IM, and 2.1× on DRISA.

Fig. 4 shows DUE and SDC probabilities under transient faults. PIM-SUM reduces DUE by over 170× under nominal error rates and maintains a 22× advantage under high-stress conditions. For SDC, it achieves up to 1778× reduction compared to ECC-PIM, with at least 220× under aggressive faults. Results confirm strong multi-bit correction and high resilience at scale.

VI. Conclusion

In this work, we propose PIM-SUM, a scalable and energy-efficient in-DRAM summation primitive tailored for embedding aggregation in large-scale recommendation systems. PIM-SUM introduces a popcount-based reduction strategy and integrates a lightweight Reed–Solomon-inspired error correction mechanism. Experimental results show that PIM-SUM consistently outperforms prior baselines, achieving up to 5.14× inference speedup.

References

[1] L. Ke *et al.*, "Recnmp: Accelerating personalized recommendation with near-memory processing," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, 2020, pp. 790–803. DOI: 10.1109/ISCA45697.2020.00070.

[2] W. Shalaby *et al.*, "Help me find a job: A graph-based approach for job recommendation at scale," in *2017 IEEE international conference on big data (big data)*, IEEE, 2017, pp. 1544–1553.

[3] C. Pei *et al.*, "Personalized re-ranking for recommendation," in *Proceedings of the 13th ACM conference on recommender systems*, 2019, pp. 3–11.

[4] K. Asifuzzaman *et al.*, "A survey on processing-in-memory techniques: Advances and challenges," *Memories-Materials, Devices, Circuits and Systems*, vol. 4, p. 100 022, 2023.

[5] V. Seshadri *et al.*, "Ambit: In-memory accelerator for bulk bitwise operations using commodity dram technology," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, 2017, pp. 273–287.

[6] N. Hajinazar *et al.*, "Simdram: A framework for bit-serial simd processing using dram," in *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2021, pp. 329–345.

[7] S. Li *et al.*, "Pinatubo: A processing-in-memory architecture for bulk bitwise operations in emerging non-volatile memories," in *Proceedings of the 53rd Annual Design Automation Conference*, 2016, pp. 1–6.

[8] S. Aga *et al.*, "Compute caches," in *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, IEEE, 2017, pp. 481–492.

[9] X. Xin *et al.*, "Elp2im: Efficient and low power bitwise operation processing in dram," in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, IEEE, 2020, pp. 303–314.

[10] S. Huang *et al.*, "Xor-cim: Compute-in-memory sram architecture with embedded xor encryption," in *Proceedings of the 39th International Conference on Computer-Aided Design*, 2020, pp. 1–6.

[11] S. Lee *et al.*, "Hardware architecture and software stack for pim based on commercial dram technology: Industrial product," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, IEEE, 2021, pp. 43–56.

[12] V. Seshadri *et al.*, "Rowclone: Fast and energy-efficient in-dram bulk data copy and initialization," in *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*, 2013, pp. 185–197.

[13] JEDEC, *JESD79-5: JEDEC Standard DDR5 SDRAM*, 2020.

[14] M. Naumov *et al.*, "Deep learning recommendation model for personalization and recommendation systems," *arXiv preprint arXiv:1906.00091*, 2019.

[15] *Anime recommendations database*, https://www.kaggle.com/datasets/CooperUnion/anime-recommendations-database, Accessed: May. 14, 2025, 2023.

[16] F. M. Harper *et al.*, "The movielens datasets: History and context," *ACM Transactions on Interactive Intelligent Systems (TiiS)*, vol. 5, no. 4, pp. 1–19, 2015.

[17] *Last.fm recommendations database*, http://millionsongdataset.com/lastfm/index.html, Accessed: May. 14, 2025, 2023.

[18] J. Ni *et al.*, "Justifying recommendations using distantly-labeled reviews and fine-grained aspects," in *Proceedings of the Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, 2019, pp. 188–197.

[19] H. Luo *et al.*, "Ramulator 2.0: A modern, modular, and extensible dram simulator," *IEEE Computer Architecture Letters*, vol. 23, no. 1, pp. 112–116, 2023.