PyGim: An Efficient Graph Neural Network Library for Real Processing-In-Memory Architectures

Christina Giannoula
University of Toronto, Canada
ETH Zürich, Switzerland
Vector Institute, Canada
CentML, Canada
christina.giann@gmail.com

Peiming Yang
University of Toronto, Canada
yang44901@gmail.com

Ivan Fernandez
Barcelona Supercomputing Center, Spain
Universitat Politècnica de Catalunya, Spain
ETH Zürich, Switzerland
ivanferveg@gmail.com

Jiacheng Yang
University of Toronto, Canada
Vector Institute, Canada
jiacheng.yang@mail.utoronto.ca

Sankeerth Durvasula
University of Toronto, Canada
Vector Institute, Canada
sankeerth.durvasula@mail.utoronto.ca

Yu Xin Li
University of Toronto, Canada
lyx.li@mail.utoronto.ca

Mohammad Sadrosadati
ETH Zürich, Switzerland
m.sadr89@gmail.com

Juan Gomez Luna
NVIDIA, Switzerland
el1goluj@gmail.com

Onur Mutlu
ETH Zürich, Switzerland
omutlu@gmail.com

Gennady Pekhimenko
University of Toronto, Canada
Vector Institute, Canada
CentML, Canada
pekhimenko@cs.toronto.edu

Abstract

Graph Neural Networks (GNNs) [1–4] have emerged as state-of-the-art Machine Learning (ML) models to depict dependent relations in graph-structure data, providing high accuracy in vertex classification and link (edge) prediction tasks [5–8]. Thus, they have been adopted to many real-world applications, including point-cloud analysis [9], recommendation systems [10], social network analysis [11], and drug discovery [12]. GNNs comprise a few layers, and each layer consists of two steps: *aggregation* and *combination*. The former aggregates the input feature vectors of the neighboring vertices for each vertex in the graph via a permutation-invariant operator (e.g., average). The latter processes the aggregated vectors of all vertices through a small neural network (e.g., a multilayer perceptron [13]) to produce the output feature vectors, which will be fed as input feature vectors to the next layer.

The key operators of combination are dense matrix matrix multiplications (GEMMs), while aggregation degenerates to a Sparse Matrix Matrix Multiplication (SpMM) kernel, processing the graph data that is represented as a sparse matrix [14–16]. We profile the GNN execution in a high-end GPU system [17, 18], and find that aggregation dominates execution time exhibiting high memory intensity, while combination is compute intensive. The compute-intensive combination fits to be executed in processor-centric systems (CPUs or GPUs). However, aggregation is significantly bottlenecked by data movement between memory and processors in such systems, since SpMM is typically memory-bandwidth-bound in CPUs and GPUs [14, 15, 19].

A promising way to alleviate the data movement cost is Processing-In-Memory (PIM) [17, 20–55] computing paradigm. PIM enables computation to be performed close to the application data by equipping memory chips with processing capabilities (in-memory processors). To provide significantly higher memory bandwidth for the in-memory processors than standard DRAM modules, manufacturers have commercialized *near-bank* PIM designs [21]. Near-bank PIM memory modules tightly couple a PIM core with one (or a few) DRAM bank, exploiting bank-level parallelism to expose the high aggregated on-chip memory bandwidth of standard DRAM to processors. A real PIM system supports multiple near-bank PIM memory modules, which are connected to a CPU or GPU, henceforth referred to as *Host*. The UPMEM PIM architecture [21] is the first PIM system to become commercially available. HBM-PIM [56] and AiM [20] are near-bank PIM systems that have been prototyped and evaluated in real systems.

A few works [15, 16, 57] propose hybrid Host-PIM accelerators for GNNs. However, none of them considers real-world PIM systems. These works design new microarchitectures for *near-rank* PIM systems, i.e., accelerator cores are placed at each rank of memory modules. Near-rank PIM designs have not been commercialized yet, and are not always able to provide significantly higher memory bandwidth for processors than standard DRAM [41, 56]. In the software level, these works have simple *fixed* parallelization strategies for GNN aggregation in PIM cores, which would cause out-of-memory errors for medium-/large-size graphs or achieve

SIGMETRICS Abstracts '25, Stony Brook, NY, USA.

ACM ISBN 979-8-4007-1593-8/2025/06
https://doi.org/10.1145/3726854.3727310

very low performance in *real* near-bank PIM systems, as we explain in our full paper [17, 18]. Moreover, these works use software emulators for their evaluations (not a real PIM system).

Our **goal** in this work is to efficiently map GNNs on near-bank PIM systems and quantify the potential of *real* PIM architectures in GNN executions. Efficiently executing GNNs in real PIM systems encounters three key challenges. 1) GNN execution has repeated compute-intensive (combination) and memory-intensive (aggregation) kernels. On the one hand, executing both types of kernels in PIM cores would incur high performance overheads in combination, since PIM cores are low-area and low-power cores with relatively low computation capabilities [26, 41, 56]. On the other hand, executing combination on Host cores and aggregation on PIM cores, respectively, necessitates minimizing the overheads of passing the output result of one kernel as input to the next kernel. 2) Real-world graphs exhibit diverse characteristics, e.g., the average, min or max vertex neighboring degrees vary across different graphs. Therefore, as discussed in prior works [24, 58–61], the execution behavior of sparse kernels, such as the SpMV/SpMM, depends on the particular characteristics of the input given, and there is no typically one-size-fits-all parallelization solution that performs best across various inputs [24]. 3) Programming a real near-bank PIM system for a high-level application is a hard task [31, 62, 63], since software stacks for PIM systems are still in an early stage. Thus, ML programmers may need to distribute data of GNNs across thousands of DRAM banks in a fine-grained and careful way, have expertise of the PIM hardware [31, 63] and/or program the PIM cores using low-level APIs [26, 31].

To address the aforementioned challenges, we design PyGim [64], a high-level ML library to efficiently execute GNNs in real PIM systems. PyGim provides high system performance in *real* Host-PIM executions of GNNs, and bridges the gap between ML engineers, who prefer high-level programming interfaces (e.g., Python), and real PIM systems, that typically provide complex and low-level APIs and may need deep knowledge of PIM hardware.

PyGim co-designs a **Cooperative Acceleration (CoA)** model with a novel **Parallelism Fusion (PaF)** method. CoA runs heterogeneous kernels to the best-fit underlying hardware: the processor-centric Host (CPU/GPU) system executes the compute-intensive GNN combination, and the memory-centric PIM system executes the memory-intensive aggregation. PaF serves a dual purpose: it (i) strives a balance between computation and data transfer costs in GNN aggregation executed in PIM cores, minimizing the overheads of passing the output result of combination as input to aggregation and vice versa, and (ii) provides various parallelization techniques to cover many real-world graphs with diverse characteristics. Specifically, in GNN aggregation, we enable three parallelism levels on the hardware PIM side and, at each level, provide different parallelization techniques on the software side. 1) We group the available PIM cores of the system in clusters, and design *edge-* and *feature-level* parallelism *across PIM clusters*. 2) We enable *vertex-* or *edge-level* parallelism across cores *within PIM cluster*. 3) We employ either *vertex-* or *edge-level* parallelism across threads *within a PIM core*. The technique of the first parallelism level reduces data transfer overheads to/from PIM memory modules, thus reducing costs when passing the output of one GNN operator as input to the next one. The techniques of the second and third parallelism levels enable load balancing schemes that provide high compute balance across low-power PIM cores and across threads of a PIM core. PaF enables various GNN aggregation configurations and load balancing strategies, by configuring the number of PIM cores per cluster, vertex- or edge-level parallelism within a PIM cluster or within a PIM core, such that to efficiently support diverse real-world graphs.

We design PyGim to adapt to the graph's characteristics with minimal programmer intervention. We integrate in PyGim a **lightweight tuner** that predicts the best-performing PaF aggregation configuration based on the particular characteristics of the input graph. PyGim's tuner employs effective performance models to estimate performance of different GNN aggregation configurations in PIM systems at low cost. This way, we automate the selection of the PyGim PaF configuration and eliminate the need for manual programmer intervention, while also providing high system performance. We develop a PIM backend for our optimized implementations and expose them with a **handy Python API** [17, 18], so that programmers can easily use them. We integrate our API with PyTorch [65] (it can be integrated to other frameworks [66–69]) to support either CPU or GPU as the Host (GPU-PIM systems are expected to be commercialized) in GNN PIM-based executions. PyGim supports two widely-used compression formats for real-world graphs. To our knowledge, PyGim is *the first easy-to-use and high-level ML library to deploy GNN models in real PIM systems*, and is available as open-source to enable the use of PIM systems in GNNs.

We comprehensively characterize GNN execution on the UPMEM PIM system, the first real-world PIM architecture, which has 16 PIM DIMMs with 1992 PIM cores connected to Host CPU. We evaluate our techniques in terms of scalability, data transfer costs, aggregation kernel and inference performance and energy using various real-world graphs and GNN models. We compare PyGim over prior state-of-the-art PIM-based works for GNNs and show that it achieves significantly higher performance by on average 4.38× (up to 7.20×) and higher energy efficiency by on average 2.86× (up to 3.68×) in GNN inference. PyGim improves GNN inference performance and energy efficiency over the state-of-the-art PyTorch scheme running on Host by on average 3.04× (up to 3.44×) and 1.55× (up to 1.75×), respectively. Moreover, PyGim achieves on average 11.6× higher resource utilization on PIM system than that of PyTorch's backend on GPU systems, which is an optimized CUDA implementation from pytorch_sparse library [70]. This means that PyGim uses the PIM system more effectively than PyTorch's backend library uses the GPU system. Our extensive study provides recommendations for PIM hardware, systems and software.

For more information about the open-source PyGim software package [64], our thorough characterization on the GNN PIM execution, results, and insights, we refer the reader to the full version of the paper [17, 18]. We hope that our parallelization strategies for GNNs, in-depth PIM analysis, and open-source library will enable further research on optimizing GNNs and other sparse ML models in memory-centric computing systems, and will enlighten architects and system designers in the development of future memory-centric computing systems. The PyGim software package is publicly and freely available at https://github.com/CMU-SAFARI/PyGim.

CCS Concepts

• **General and reference** → **Performance**; **Design**; **Evaluation**; **Experimentation**; • **Computer systems organization** → **Architectures**; • **Hardware** → **Power and energy**;

Keywords

machine learning, graph neural networks, sparse matrix-matrix multiplication, library, framework, processing-in-memory, near-data processing, memory systems, data movement, DRAM, benchmarking, real-system characterization, workload characterization

ACM Reference Format:
Christina Giannoula, Peiming Yang, Ivan Fernandez, Jiacheng Yang, Sankeerth Durvasula, Yu Xin Li, Mohammad Sadrosadati, Juan Gomez Luna, Onur Mutlu, and Gennady Pekhimenko. 2025. PyGim: An Efficient Graph Neural Network Library for Real Processing-In-Memory Architectures. In *Abstracts of the 2025 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS Abstracts '25), June 9–13, 2025, Stony Brook, NY, USA.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3726854.3727310

References

[1] Thomas N Kipf et al. 2016. Semi-Supervised Classification with Graph Convolutional Networks. *arXiv* (2016).
[2] Keyulu Xu, et al. 2018. How Powerful Are Graph Neural Networks? *arXiv* (2018).
[3] Will Hamilton, et al. 2017. Inductive Representation Learning on Large Graphs. *NIPS* 30 (2017).
[4] D. Zheng, et al. 2020. DistDGL: Distributed Graph Neural Network Training for Billion-Scale Graphs. In *IA3*.
[5] Chao Shang, et al. 2021. Discrete Graph Structure Learning for Forecasting Multiple Time Series. In *ICLR*.
[6] John Thorpe, et al. 2021. Dorylus: Affordable, Scalable, and Accurate GNN Training with Distributed CPU Servers and Serverless Threads. In *OSDI*.
[7] Sameh K Mohamed, et al. 2020. Discovering Protein Drug Targets Using Knowledge Graph Embeddings. *Bioinformatics* (2020).
[8] Rex Ying, et al. 2018. Graph Convolutional Neural Networks for Web-Scale Recommender Systems. In *SIGKDD*.
[9] Guocheng Qian, et al. 2021. Pu-GCN: Point Cloud Upsampling Using Graph Convolutional Networks. In *CVPR*.
[10] Shu Wu, et al. 2019. Session-Based Recommendation with Graph Neural Networks. In *AAAI*.
[11] Wenqi Fan, et al. 2019. Graph Neural Networks for Social Recommendation. In *The World Wide Web Conference*.
[12] Jonathan M Stokes, et al. 2020. A Deep Learning Approach to Antibiotic Discovery. *Cell* (2020).
[13] Christopher M Bishop. 1995. *Neural Networks for Pattern Recognition*. Oxford University Press.
[14] Zhangxiaowen Gong, et al. 2022. Graphite: Optimizing Graph Neural Networks on CPUs through Cooperative Software-Hardware Techniques. In *ISCA*.
[15] Sungmin Yun, et al. 2023. GraNDe: Efficient Near-Data Processing Architecture for Graph Neural Networks. *IEEE Trans. Comput.* (2023).
[16] Zhe Zhou, et al. 2022. Gnnear: Accelerating Full-Batch Training of Graph Neural Networks with Near-Memory Processing. In *PACT*.
[17] Christina Giannoula, et al. 2024. Accelerating Graph Neural Networks on Real Processing-In-Memory Systems. https://arxiv.org/abs/2402.16731
[18] Christina Giannoula, et al. 2024. PyGim: An Efficient Graph Neural Network Library for Real Processing-In-Memory Architectures. *Proc. ACM Meas. Anal. Comput. Syst.* (2024). https://doi.org/10.1145/3700434
[19] Zhixiang Gu, et al. 2020. Bandwidth Optimized Parallel Algorithms for Sparse Matrix-Matrix Multiplication Using Propagation Blocking. In *SPAA*.
[20] Seongju Lee, et al. 2022. A 1ynm 1.25 V 8GB, 16GB/s/Pin GDDR6-Based Accelerator-in-Memory Supporting 1Tflops MAC Operation and Various Activation Functions for Deep-Learning Applications. In *ISSCC*.
[21] F. Devaux. 2019. The True Processing In Memory Accelerator. In *Hot Chips*.
[22] Onur Mutlu, et al. 2021. A Modern Primer on Processing in Memory. In *Emerging Computing: From Devices to Systems - Looking Beyond Moore and Von Neumann*. https://arxiv.org/pdf/2012.03112.pdf
[23] Juan Gómez-Luna, et al. 2022. Benchmarking a New Paradigm: Experimental Analysis and Characterization of a Real Processing-in-Memory System. *IEEE Access* (2022).
[24] Christina Giannoula, et al. 2022. SparseP: Towards Efficient Sparse Matrix Vector Multiplication on Real Processing-in-Memory Architectures. *POMACS* (2022).
[25] Safaa Diab, et al. 2023. A Framework for High-Throughput Sequence Alignment Using Real Processing-in-Memory Systems. *Bioinformatics* (2023).
[26] Juan Gómez-Luna, et al. 2023. Evaluating Machine Learning Workloads on Memory-Centric Computing Systems. In *ISPASS*.
[27] Chaemin Lim, et al. 2023. Design and Analysis of a Processing-in-DIMM Join Algorithm: A Case Study with UPMEM DIMMs. *Proc. ACM Manag. Data* (2023).
[28] Maurus Item, et al. 2023. TransPimLib: Efficient Transcendental Functions for Processing-in-Memory Systems. In *ISPASS*.
[29] Prangon Das, et al. 2022. Implementation and Evaluation of Deep Neural Networks in Commercially Available Processing in Memory Hardware. In *SOCC*.
[30] Muhammad Attahir Jibril, et al. 2024. Accelerating Aggregation Using a Real Processing-in-Memory System. In *ICDE*.
[31] Jinfan Chen, et al. 2023. SimplePIM: A Software Framework for Productive and Efficient Processing-in-Memory. In *PACT*.
[32] Yongwon Shin, et al. 2023. PIMFlow: Compiler and Runtime Support for CNN Models on Processing-in-Memory DRAM. In *CGO*.
[33] Steve Rhyner, et al. 2024. Analysis of Distributed Optimization Algorithms on a Real Processing-In-Memory System. In *PACT*.
[34] Christina Giannoula, et al. 2021. SynCron: Efficient Synchronization Support for Near-Data-Processing Architectures. In *HPCA*.
[35] Ivan Fernandez, et al. 2024. MATSA: An MRAM-Based Energy-Efficient Accelerator for Time Series Analysis. *IEEE Access* (2024).
[36] Benjamin Y. Cho, et al. 2020. Near Data Acceleration with Concurrent Host Access. In *ISCA*.
[37] Youngeun Kwon, et al. 2019. TensorDIMM: A Practical Near-Memory Processing Architecture for Embeddings and Tensor Operations in Deep Learning. In *MICRO*.
[38] Mingyu Gao, et al. 2015. Practical Near-Data Processing for In-Memory Analytics Frameworks. In *PACT*.
[39] Ivan Fernandez, et al. 2020. NATSA: A Near-Data Processing Accelerator for Time Series Analysis. In *ICCD*.
[40] Liu Ke, et al. 2020. RecNMP: Accelerating Personalized Recommendation with Near-Memory Processing. In *ISCA*.
[41] Hadi Asghari-Moghaddam, et al. 2016. Chameleon: Versatile and Practical Near-DRAM Acceleration Architecture for Large Memory Systems. In *MICRO*.
[42] Junwhan Ahn, et al. 2015. A Scalable Processing-In-Memory Accelerator for Parallel Graph Processing. In *ISCA*.
[43] Kevin Hsieh, et al. 2016. Accelerating Pointer Chasing in 3D-stacked Memory: Challenges, Mechanisms, Evaluation. In *ICCD*.
[44] Youwei Zhuo, et al. 2019. GraphQ: Scalable PIM-based Graph Processing. In *MICRO*.
[45] Amirali Boroumand, et al. 2018. Google Workloads for Consumer Devices: Mitigating Data Movement Bottlenecks. In *ASPLOS*.
[46] Juan Gómez-Luna, et al. 2021. Benchmarking a New Paradigm: An Experimental Analysis of a Real Processing-in-Memory Architecture. In *CoRR*. https://arxiv.org/abs/2105.03814
[47] Mario Drumond, et al. 2017. The Mondrian Data Engine. In *ISCA*.
[48] Jiawen Liu, et al. 2018. Processing-in-Memory for Energy-Efficient Neural Network Training: A Heterogeneous Approach. In *MICRO*.
[49] Zhiyu Liu, et al. 2017. Concurrent Data Structures for Near-Memory Computing. In *SPAA*.
[50] Jiwon Choe, et al. 2019. Concurrent Data Structures with Near-Data-Processing: An Architecture-Aware Implementation. In *SPAA*.
[51] Maya Gokhale, et al. 2015. Near Memory Data Structure Rearrangement. In *MEMSYS*.
[52] Mingxing Zhang, et al. 2018. GraphP: Reducing Communication for PIM-Based Graph Processing with Efficient Data Partition. In *HPCA*.
[53] Onur Mutlu, et al. 2021. A Modern Primer on Processing in Memory. *Emerging Computing: From Devices to Systems - Looking Beyond Moore and Von Neumann* (2021).
[54] Saugata Ghose, et al. 2019. Processing-in-Memory: A Workload-Driven Perspective. In *IBM JRD*.
[55] Onur Mutlu, et al. 2019. Processing Data Where It Makes Sense: Enabling In-Memory Computation. In *MICPRO*.
[56] Sukhan Lee, et al. 2021. Hardware Architecture and Software Stack for PIM Based on Commercial DRAM Technology: Industrial Product. In *ISCA*.
[57] Teng Tian, et al. 2022. G-NMP: Accelerating Graph Neural Networks with DIMM-Based Near-Memory Processing. *Journal of Systems Architecture* (2022).
[58] Konstantinos Kanellopoulos, et al. 2019. Smash: Co-Designing Software Compression and Hardware-Accelerated Indexing for Efficient Sparse Matrix Operations. In *MICRO*.
[59] Damitha Lenadora, et al. 2024. SENSEi: Input-Sensitive Compilation for Accelerating GNNs. In *arXiv*.
[60] Zihao Ye, et al. 2023. SparseTIR: Composable Abstractions for Sparse Compilation in Deep Learning. In *ASPLOS*.
[61] Foteini Strati, et al. 2019. An Adaptive Concurrent Priority Queue for NUMA Architectures. In *International Conference on Computing Frontiers*.
[62] Bongjoon Hyun, et al. 2024. Pathfinding Future PIM Architectures by Demystifying a Commercial PIM Technology. In *HPCA*.
[63] Gilbert Jonatan, et al. 2024. Scalability Limitations of Processing-in-Memory Using Real System Evaluations. *Proc. ACM Meas. Anal. Comput. Syst.* (2024).
[64] SAFARI Research Group. 2022. PyGim Software Package. https://github.com/CMU-SAFARI/PyGim
[65] Adam Paszke, et al. 2019. Pytorch: An Imperative Style, High-Performance Deep Learning Library. *NIPS* (2019).
[66] Martín Abadi, et al. 2016. Tensorflow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. *arXiv* (2016).
[67] Antonio Gulli et al. 2017. *Deep Learning with Keras*. Packt Publishing Ltd.
[68] Tianqi Chen, et al. 2015. Mxnet: A Flexible and Efficient Machine Learning Library for Heterogeneous Distributed Systems. *arXiv* (2015).
[69] Yangqing Jia, et al. 2014. Caffe: Convolutional Architecture for Fast Feature Embedding. In *Proceedings of the 22nd ACM International Conference on Multimedia*.
[70] Matthias Fey et al. 2019. Fast Graph Representation Learning with PyTorch Geometric. In *ICLR*.