

HyPIM: Accelerating Hyperbolic Machine Learning via Processing-in-Memory

Jinlin Wu
School of Computing
Illinois Institute of Technology
Chicago, USA
jwu141@hawk.illinoistech.edu

Ferit Ozdaban
School of Computing
Illinois Institute of Technology
Chicago, USA
fozdaban@hawk.illinoistech.edu

Selvanathan Sadhasivam
School of Computing
Illinois Institute of Technology
Chicago, USA
ssadhasivam@hawk.illinoistech.edu

Xiaoyang Lu
School of Computing
Illinois Institute of Technology
Chicago, USA
xlu40@illinoistech.edu

Yunhui Guo
Department of Computer Science
University of Texas at Dallas
Dallas, USA
yunhui.guo@utdallas.edu

Minxuan Zhou
School of Computing
Illinois Institute of Technology
Chicago, USA
mzhou26@illinoistech.edu

Abstract—Recent research has demonstrated that hyperbolic machine learning, which utilizes hyperbolic geometry to represent data and define arithmetic operations, can substantially enhance the efficiency and accuracy of machine learning tasks on hierarchical data. Unfortunately, the existing machine learning accelerators cannot efficiently support hyperbolic operations, which require memory-intensive data shuffling and reduction operations, significantly degrading the utilization of arithmetic-intensive hardware. In this work, we propose HyPIM, a novel hardware-software co-design that exploits processing-in-memory (PIM) for accelerating memory-intensive hyperbolic operations in a heterogeneous system. Our experiments on real-world hyperbolic machine learning models show that HyPIM significantly improves the performance of state-of-the-art GPUs by $3\times$.

Index Terms—hyperbolic machine learning, processing-in-memory, machine learning acceleration, heterogeneous architecture

I. INTRODUCTION

Many real-world datasets, from knowledge graphs and text ontologies to biological taxonomies, possess inherent hierarchical structures [1]. Euclidean space expands linearly and assumes uniform curvature, preventing it from faithfully capturing the exponential growth of nodes or the varying distances between different hierarchy levels, often resulting in severe distortion when embedding tree-like structures [2]. Hyperbolic space, on the other hand, has constant negative curvature and exponentially expanding volume, enabling compact and low-distortion representations of hierarchical data [3]. Recent research has explored hyperbolic geometry as a new geometric foundation for machine learning and shown remarkable performance in modeling knowledge graphs, natural language hierarchies, and visual recognition tasks [4]–[6].

Despite these algorithmic advances, hyperbolic machine learning introduces fundamentally different computational patterns from conventional Euclidean machine learning. In partic-

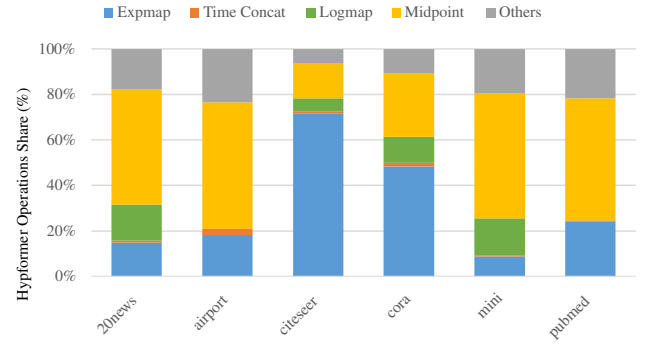


Fig. 1. Operation breakdown on Hypformer [6].

ular, most modern hyperbolic models adopt the Lorentz model, which represents an n -dimensional hyperbolic manifold as a hyperboloid embedded in $(n + 1)$ -dimensional Minkowski space. This formulation enables efficient computation of distances, inner products, and geodesics through Lorentzian algebra, but its core operations involve iterative reductions, normalization, and transcendental functions, all of which are highly memory-intensive and inefficient to execute on GPUs.

Fig. 1 shows our simulated inference latency of the hyperbolic Transformer model [6] over several representative datasets. The results show that over 80% of the total execution time is spent on hyperbolic operations rather than on the matrix-multiplication kernels that dominate conventional machine learning workloads. We observe that the cost is not concentrated in a single primitive; instead, operations such as exponential and logarithmic maps, Lorentzian distance evaluations, and curvature-dependent normalizations collectively account for a substantial fraction of end-to-end latency. These operations exhibit low arithmetic intensity and frequent reductions, resulting in poor utilization of GPU compute units

TABLE I
HYPERBOLIC OPERATORS IN THE LORENTZ MODEL. HERE, x_t DENOTES THE TIME COORDINATE, $\mathbf{x}$ DENOTES THE SPACE COORDINATE, $\mathbf{v}$ IS A TANGENT VECTOR AT THE ORIGIN, AND k IS THE CURVATURE

Operation	Lorentz Model
Time concat	$x_t = \ \mathbf{x}\ _2$
Exponential map	$\frac{1}{\sqrt{-k}} \left[\cosh(\sqrt{-k}\ \mathbf{x}\), \sinh(\sqrt{-k}\ \mathbf{x}\) \frac{\mathbf{x}}{\ \mathbf{x}\ } \right]$
Logarithmic map	$\frac{1}{\sqrt{-k}} \operatorname{arcosh}(x_t \sqrt{-k}) \frac{\mathbf{x}}{\ \mathbf{x}\ }$
Midpoint	$\frac{(x_t + y_t, \mathbf{x} + \mathbf{y})}{\sqrt{2(1 - k(-x_t y_t + \mathbf{x} \cdot \mathbf{y}))}}$
Parallel transport	$\left(\frac{\mathbf{v} \cdot \mathbf{x}}{\sqrt{-k}\ \mathbf{x}\ _2}, \mathbf{v} - \frac{\mathbf{v} \cdot \mathbf{x}}{\ \mathbf{x}\ _2^2} \mathbf{x} \right)$

that are heavily optimized for dense tensor arithmetic.

Prior studies have shown that even extensive kernel-level optimizations on GPUs cannot fully overcome the I/O and bandwidth bottlenecks of memory-intensive operators [7], [8]. To address these limitations, recent Processing-in-Memory (PIM)-based systems execute memory-intensive primitives directly near or inside DRAM, achieving substantial improvements in throughput and energy efficiency while reducing dependence on costly GPUs [9]. However, existing PIM accelerators [7]–[19] cannot effectively accelerate hyperbolic machine learning models due to the lack of hardware functionality and software support for hyperbolic-specific data structures and operations.

In this work, we first identify a fundamental compute–memory imbalance in modern Lorentz-model hyperbolic networks: Euclidean layers are compute-intensive and GPU-friendly, whereas hyperbolic operators are memory-bound and severely underutilize GPUs. Based on these observations, we propose HyPIM, a heterogeneous GPU–PIM co-design that processes compute-intensive linear layers on the GPU while offloading memory-intensive hyperbolic primitives to HBM-integrated PIM. We develop a PIM architecture based on state-of-the-art high-bandwidth memory (HBM3). The proposed PIM accelerator employs lightweight near-bank GEMV engines with channel-level transcendental units to support both reductions and transcendental operations efficiently. Furthermore, we design an optimized hyperbolic-friendly data layout and an in-flight wavefront scheduler that overlaps GEMV streaming with transcendental execution for hyperbolic operators to maximize internal bandwidth utilization of the PIM architecture. We conduct a comprehensive evaluation of state-of-the-art hyperbolic neural networks on vision and graph datasets to validate the efficiency of HyPIM. Our experiments show that HyPIM provides up to 80 $\times$ speedup for hyperbolic operators and more than 3 $\times$ end-to-end speedup over an A100 GPU baseline.

II. BACKGROUND AND MOTIVATION

A. Hyperbolic Machine Learning Models

Hyperbolic geometry provides a natural embedding space for hierarchical datasets due to its exponential expansion property and negative curvature. Many real-world data structures,

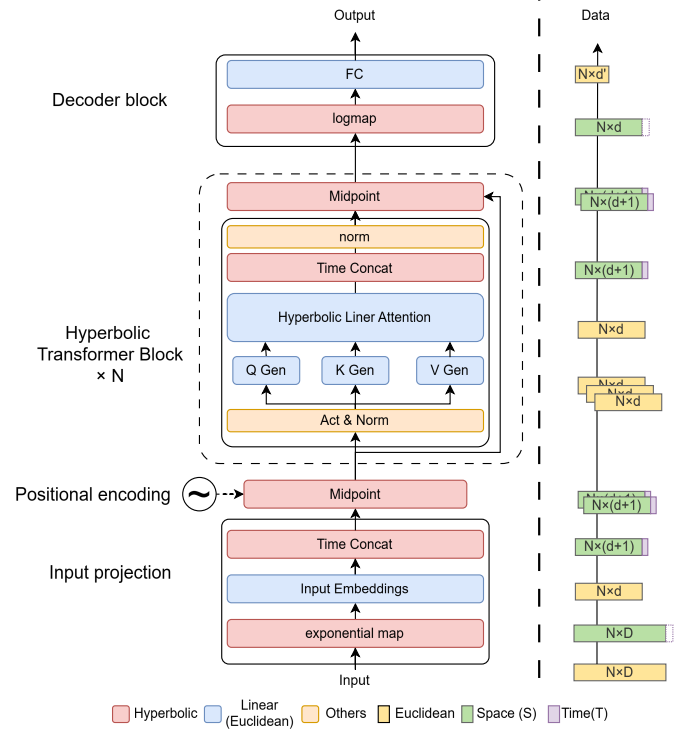


Fig. 2. Framework of Hypformer [6].

such as taxonomies and knowledge graphs, can be represented more compactly and with less distortion in hyperbolic space than in Euclidean space. Recent studies employ the Lorentz model, which transforms each point to $(d + 1)$ -dimensional Minkowski space, to implement hyperbolic neural networks (NNs), including Lorentz-based CNNs [5], GNNs [20], and Transformers [6]. These hyperbolic NNs follow a common computational pattern: interleaving standard Euclidean layers with geometry-preserving transformations defined on the Lorentz model, as shown in Fig. 2.

Within this hybrid computation framework, hyperbolic models rely on a small set of geometry-preserving operators shown in Table I that enable stable transitions between Euclidean layers and the Lorentz manifold. The **exponential map** (expmap) and **logarithmic map** (logmap) provide the fundamental mechanism for moving representations between the manifold and its tangent space. Logmap converts Lorentz points $x \in \mathbb{R}^{B \times N \times (d+1)}$ into Euclidean tangent vectors $v \in \mathbb{R}^{B \times N \times (d)}$ used by linear or convolutional layers, while expmap maps these vectors back onto the hyperboloid by restoring the time coordinate with hyperbolic functions. **Midpoint** operators aggregate multiple Lorentz points into a single geometry-consistent representative by computing a Minkowski-weighted average and renormalizing it onto the manifold. **Time-concatenation** reconstructs the Lorentz time component (i.e., the additional dimension in $(d + 1)$) after Euclidean updates to the space coordinates, ensuring the hyperbolic constraint is satisfied. **Parallel transport** moves tangent vectors between different tangent spaces while preserving their Lorentz norm and direction, enabling consistent

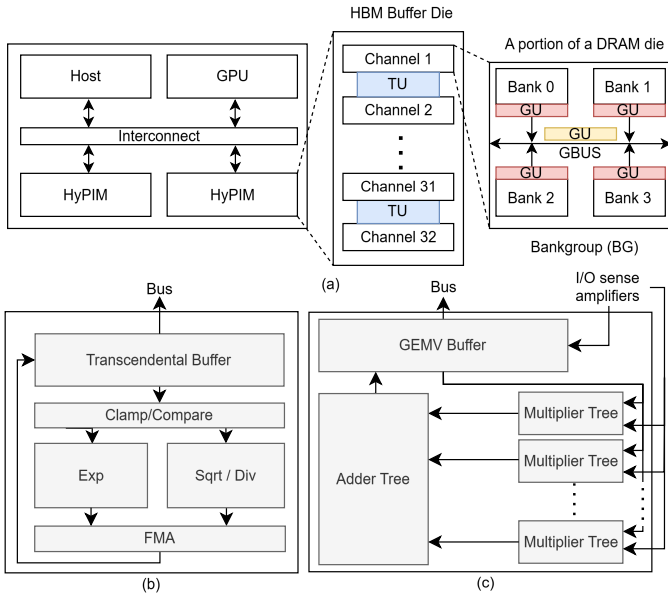


Fig. 3. (a) Overall HyPIM architecture. Red blocks indicate the bank-level GU layout, while yellow blocks represent the bank-group-level GU layout. (b) and (c) Details of the TU and GU.

vector updates across multiple layers.

These hyperbolic operators involve transcendental functions, long vector reductions, and geometry-enforcing normalizations, resulting in very low arithmetic intensity. Because each step performs only a small amount of computation per byte of data loaded, the operators are inherently memory-bound, preventing GPUs from fully leveraging their computational throughput. As shown in Fig. 1, these operators often become the dominant performance bottleneck in hyperbolic models and therefore motivate processing-in-memory acceleration for hyperbolic machine learning.

B. DRAM-Based Processing-In-Memory

The increasing computational intensity of modern applications has shown a continuing imbalance between processor throughput and memory bandwidth. Processing-in-memory places computation near or inside DRAM to exploit internal bandwidth and reduce data movement. We target high-bandwidth-memory-based PIM, whose stack comprises a buffer die and multiple memory dies organized into pseudo-channels, bank groups, and banks connected through TSVs. This hierarchy provides high aggregate bandwidth and exposes bank-level parallelism for memory-bound operators [7], [11].

III. HYPIM DESIGN

As illustrated in Section II-A, the hybrid computational structure of hyperbolic NNs motivates us to investigate an efficient heterogeneous computing platform. Fig. 3 shows the overall architecture of our proposed system, which combines GPUs with HyPIM, an HBM3-based processing-in-memory (PIM) accelerator for hyperbolic operations. The division of labor is guided by arithmetic intensity and reuse: compute-dense operators with high data reuse, such as GEMM, remain

on the GPU, while memory-bound hyperbolic operations, such as linear contractions and reductions, execute inside HBM. PIM-GPU communication uses an asynchronous DMA path (e.g., NVLink/PCIe) with double-buffered handoff: while PIM writes one buffer, a dedicated copy engine streams the previous buffer to the GPU without stalling computation. Under multi-batch execution, this design overlaps transfer latency with the execution of adjacent batches.

A. HyPIM Hardware Units

HyPIM integrates hardware units for two memory-intensive kernel families that appear repeatedly in hyperbolic machine learning models: (1) GEMV operations that compute reductions or dot products over high-dimensional hyperbolic features; (2) transcendental functions to scale and normalize space-like vectors and construct the correct time-like coordinate.

1) *GEMV Unit (GU)*: The first kernel family consists of elementwise multiplications and reductions over long hyperbolic feature vectors. To accelerate these operations near memory, we instantiate a GEMV unit at each near-bank or bank-group position, depending on the placement mode. Each GEMV unit streams column data directly from the HBM array to process element-wise multiplications and reductions. The unit contains 16 FP16 multipliers and 16 FP16 adders, which are provisioned to saturate the 256-bit internal data link of HBM. Each lane of the unit receives one 16-bit datum from the bank and another 16-bit operand from the GEMV buffer (GB). The GB is a 16×256 -bit buffer that holds input vectors and partial sums locally, so repeated accesses remain inside the unit without writing intermediate results back to and rereading them from DRAM.

2) *Transcendental Unit (TU)*: The second kernel family consists of scalar transcendental primitives required to preserve hyperbolic geometry. We introduce a narrow transcendental unit shared by two adjacent channels so that hyperbolic-specific stages can execute in situ near the data, rather than shuttling data back to the GPU. The TU handles the following primitives: clamp and saturation guards; square root; division; and exponentials. We move data from the GU to the TU for these primitives and return the results to the GU for subsequent GEMV computations. Microarchitecturally, the TU consists of a lightweight collection of lane-narrow transcendental operators together with a double-buffered Transcendental Buffer (TB) that stages inputs and outputs close to the execution units. At the front of the datapath, clamp-and-compare operations are implemented as fused min/max units to ensure numerical stability before an operand enters any transcendental pipeline. The exp/sinh/cosh primitives use standard range-reduction techniques (power-of-two exponent peeling and mantissa normalization), followed by a compact LUT-plus-polynomial evaluation path. Square root and division rely on short Newton-Raphson iterations seeded from a small LUT; for square root, the TU implements rsqrt followed by a final multiply using an internal multiply-add unit.

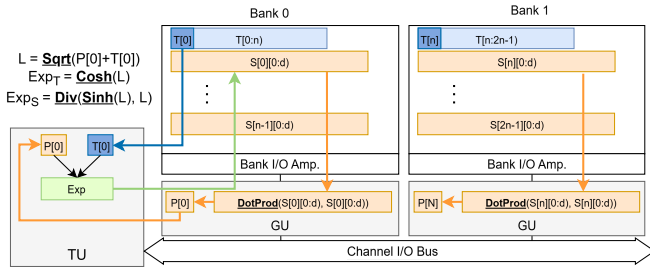


Fig. 4. Optimized data layout for hyperbolic data and expmap processing flow.

B. Unit Placement

To understand how GU placement influences contraction throughput and the interaction with the TU, we examine two alternative integration strategies for the GU within the HBM stack.

Bank-level placement integrates GU near the column decoder so that computation is physically closest to DRAM. Each bank streams its own column data and performs local reduction, exposing the highest degree of parallelism and maintaining bandwidth for linear contractions. The downside is the replication of many small engines: the total area and static power increase, and because each GU draws from a single bank, ACT/PRE latency cannot be hidden by overlapping accesses across banks within the same engine. With the TU fixed at the channel, many bank-level producers also contend for shared TU/TB bandwidth and issue slots; back-pressure at the TU tail prevents the GUs from achieving ideal throughput.

Bank-group (BG)-level placement integrates a GU next to the GBUS controller and pairs it with a local BG buffer. For contractions that naturally span multiple banks, this reduces cross-bank traffic at large tile sizes and alleviates broadcast pressure. However, the reduced number of GUs placed per physical channel (pCH) inherently limits the available compute parallelism, which in turn constrains the peak throughput relative to the bank-level design.

C. HyPIM Data Layout and Processing Flow

Hyperbolic operators introduced in Section II-A exhibit a split computational structure: vector-heavy space computations that require contiguous streaming, and scalar Lorentz-metric expressions that need fast access to a single time value. A naïve layout that mixes space and time coordinates disrupts vector locality and slows down scalar stages, reducing PIM efficiency.

This motivates our proposed data layout, which decouples space-vector streaming from scalar time access and balances work across banks, as shown in Fig. 4. All nodes are first evenly distributed across all banks (i.e., n nodes per bank), ensuring balanced bandwidth utilization and uniform PIM working sets. Within a bank, the d -dimensional space vector of each node is mapped onto consecutive columns: given a fixed 32-byte-per-column granularity, we compute the number of columns and rows required for each node’s space vector based on the bank’s column capacity. The time coordinates of all nodes—used only in scalar Lorentz-metric expressions—are

stored in one compact, contiguous address region that is independent of the space-vector layout.

For **vector-heavy space computation with extremely low arithmetic intensity**, all operators apply reductions (norms or inner products) and element-wise transformations over the d -dimensional space vector of each node. These operations stream naturally over the column-space layout in each bank to the bank-level or bank-group-level GUs, benefiting from PIM’s high in situ bandwidth. For **scalar, time-dependent control flow decoupled from vector execution**, each operator uses the time coordinate only in a small number of scalar Lorentz-metric expressions (e.g., arcosh distance, curvature scaling, and hyperbolic constraint reconstruction). Our layout isolates time values into compact, bank-local scalar arrays, enabling fast TU access via the channel I/O bus without interfering with GU vector streaming. Fig. 4 shows an example of processing expmap with the proposed data layout.

D. HyPIM Scheduling

Sequential execution of hyperbolic operators interleaves GU operations (e.g., inner products and reductions), data movement, and long-latency transcendental primitives (DIV, SQRT, and EXP), which can reduce hardware utilization. We propose an in-flight scheduling strategy to transfer tasks between the TU and GU, and an in-bank-group scheduling strategy to coordinate tasks within each bank group under BG-level placement. HyPIM’s scheduling is realized as a runtime-generated instruction stream: the runtime decomposes each node into multiple stages as fine-grained tasks, and then issues GU/TU/movement instructions according to our scheduling strategy, incurring no additional hardware overhead for scheduling.

1) In-flight scheduling: To mitigate stall bubbles caused by long-latency transcendental stages, we schedule a hyperbolic operator over a batch of nodes at the (node, stage) granularity within a single bank. The runtime maintains a task queue for each bank and keeps up to `MAX_PIPELINE_DEPTH` nodes in flight. It realizes the wavefront schedule as an instruction stream that interleaves stages across these nodes: while one node is blocked on a TU request, the runtime continues issuing GU MAC/movement instructions for other ready nodes, thereby maximizing GU utilization and overlapping TU latency and cross-buffer movement.

When a node reaches a transcendental stage, the scheduler (i) inserts a barrier before moving data from GB to TB (to seal producer state), (ii) issues a TU request tagged to this bank; the TU is shared at the channel level and arbitrates among banks in the same channel, and (iii) immediately rotates to the next node in the bank’s deque so that nonblocked nodes keep issuing fast MAC/move stages while the TU drains in the background. After the TU completes, a barrier is inserted before moving results from TB back to GB, ensuring that downstream consumers observe a consistent view. Barriers are thus aligned with stage boundaries and placed only around cross-buffer moves.

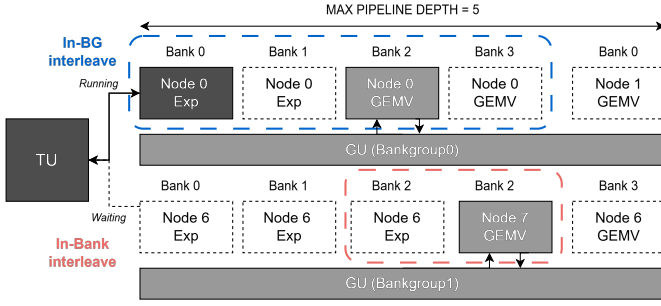


Fig. 5. Wavefront with a BG-placed GU and a channel-level TU. Within each bank, nodes advance at the (node, stage) granularity; when a transcendental stage is reached, barriers are placed around GB–TB moves, a channel-shared TU request is issued for that bank, and the scheduler proceeds to the next bank/node while the TU drains.

2) *In-bank-group scheduling*: While wavefront scheduling is bank-local, the placement of the GU at the bank-group (BG) level implies that the four banks within a BG are traversed sequentially by the GU. Meanwhile, the TU remains shared at the channel level and services TU requests across banks in the channel. To respect these scopes, we maintain a lightweight per-bank context (current node and stage) and advance banks in a deterministic per-BG order; when a bank reaches a transcendental stage, we insert the same (node, stage) barriers around GB and TB moves, tag the TU request to that bank, and continue advancing the next bank in the BG’s order while the TU drains. Thus, nonblocked banks in the BG continue issuing fast MAC/move stages, even though the GU visits banks sequentially. Fig. 5 shows in-bank and intra-BG scheduling operating concurrently on the BG-level GU.

IV. EXPERIMENTS

A. Experimental Setup

We evaluate HyPIM using a cycle-accurate simulator modified from Ramulator2 [21], extended with our PIM instruction set and support for simulating our hardware design. We simulate a heterogeneous system combining an NVIDIA A100 GPU and HyPIM, and compare against a GPU-only (A100) baseline. Following the same methodology as prior work [11], we estimate GPU execution time based on ideal computation throughput and memory throughput for a fair comparison to our simulated architecture. We model an HBM3-PIM system operating at 5.2 Gb/s with 16 pseudo-channels and 32 banks per channel, and use this configuration as the PIM-side HBM setting in all our experiments. For both bank- and bank-group-level placements, we set `MAX_PIPELINE_DEPTH` to 8.

We benchmark two representative state-of-the-art hyperbolic machine learning models: (i) *Hypformer* [6], a transformer-based model that replaces all Euclidean projections and attention operations with Lorentz-domain hyperbolic mappings; (ii) *HCNN* [5], a convolutional network that performs feature extraction entirely in the Lorentz model using hyperbolic convolutions.

We use six datasets for Hypformer: Airport, Cora, Cite-seer, and PubMed [22] are four small/medium-scale graph datasets [20]. 20 Newsgroups and Mini-ImageNet represent

TABLE II
AREA AND STATIC POWER BREAKDOWN OF GEMV UNITS AND TRANSCENDENTAL UNITS IN OUR SYSTEM

Component	Area (mm ²)	Power (mW)
GEMV Unit		
Multipliers (16 lanes)	0.004768	0.00232
Adders/reduction tree (16 lanes)	0.004454	0.001488
GEMV buffer	0.0005133	0.0307
Transcendental Unit		
FMA	0.000372	0.000181
Exp	0.001428	0.00051
Division	0.002963	0.00136
Sqrt	0.009931	0.0045
Clamp/compare	0.000072	0.000024
Transcendental buffer	0.00280446	0.86617
Bank placement total (512 GU, 8 TU)	5.125	24.650
Bank-group placement total (128 GU, 8 TU)	1.385	11.399

text and image classification tasks without an inherent graph structure [23], [24]. For HCNN, we evaluate three standard image recognition datasets—Tiny-ImageNet, CIFAR-100, and CelebA—which cover medium-scale object recognition and face-attribute prediction [25]–[27].

B. Area and Power Analysis

All logic components in the GEMV unit (GU) and the transcendental unit (TU) are described in Verilog and synthesized using the OpenROAD toolchain [28] with a 7 nm ASAP7 standard-cell library [29]. The GU consists of sixteen FP16 fused-multiply-add (FMA) lanes and a 16×256-bit local buffer. The TU integrates *SQRT/DIV*, *CLAMP*, *FMA*, and *EXP* units and a 32-KB buffer that holds the inputs and outputs of TU tasks from all 64 banks in a channel pair. We use FinCACTI [30] to model the SRAM-based GEMV and transcendental buffers.

Table II shows the area and power breakdown of GUs and TUs. Recently published HBM3 implementations [31], [32] report a DRAM-die area of 120 mm². Our hyperbolic extensions introduce less than 6% area overhead relative to a typical HBM3 die, even with aggressive bank-level placement. Since the TUs are located on the logic die, we use HotSpot 6.0 to verify that the added power does not cause the DRAM temperature to exceed 85 °C under passive cooling. The total power density is 4.97 W/cm² and does not cause thermal violations according to [33].

Fig. 7 reports the energy consumption of hyperbolic operations on the PIM and GPU, estimated using the trace-driven methodology in [9], [11]. We compute total PIM energy by mapping each PIM instruction to a set of micro-operations and accumulating their per-operation energy together with DRAM array and I/O energy based on an HBM power model. To model DRAM energy, we adopt the ACT/PRE and RD/WR cell energy values from [34]. Notably, bank-group-level placement incurs higher energy because data must traverse the bank-group MUX before GEMV computation, resulting in a longer and more energy-intensive path than in bank-level placement. This overhead becomes more significant in the HCNN model, where hyperbolic operators dominate

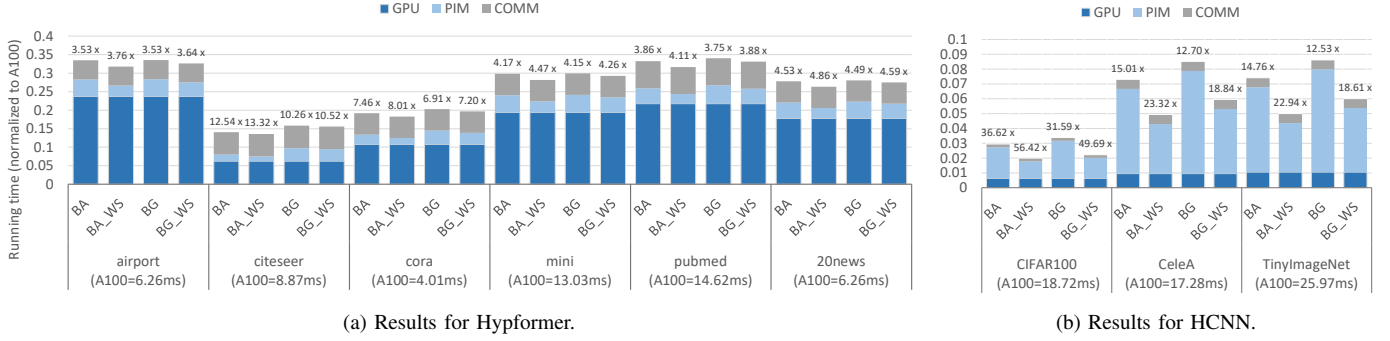


Fig. 6. End-to-end normalized running time for a single inference under four configurations: bank-level GU (BA), bank-level GU with wavefront scheduling (BA_WS), bank-group-level GU (BG), and bank-group-level GU with wavefront scheduling (BG_WS). Each bar shows the breakdown of total execution time into PIM execution, GPU execution, and GPU-PIM communication overhead (COMM).

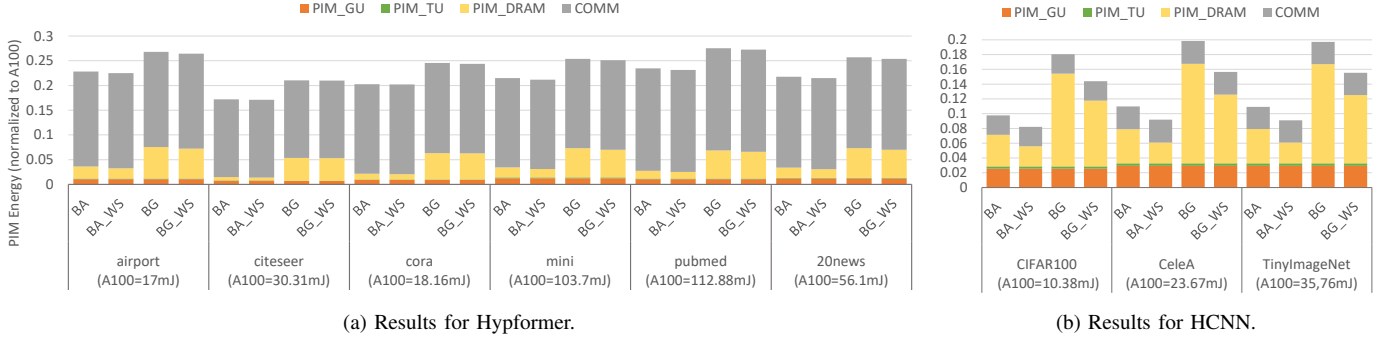


Fig. 7. Normalized energy breakdown of hyperbolic operators under the four configurations in Fig. 6.

the workflow. Cross-bank-group movement becomes a major energy contributor, making bank-level placement substantially more energy-efficient for hyperbolic workloads.

C. Overall HyPIM Performance

We first evaluate the end-to-end performance of HyPIM across all datasets and models, including both hyperbolic and non-hyperbolic layers. Fig. 6 summarizes the overall speedup of HyPIM over the A100 GPU baseline under four configurations.

a) Hypformer Speedup: HyPIM achieves a 3–13 $\times$ overall speedup on Hypformer. Citeseer exhibits the largest performance gain among all datasets. Its feature dimensionality is substantially higher than the others (3703 dimensions for 3327 nodes). On the A100 GPU, hyperbolic kernels become strongly memory-bound as the input size increases: each operation repeatedly streams a long feature vector through the memory hierarchy, and the arithmetic units remain underutilized. HyPIM leverages the high internal bandwidth and in situ computing capability of HBM-PIM, allowing long feature vectors to be processed entirely within the memory stack without repeated off-chip transfers. As a result, workloads with large input dimensionality or broad per-node computation—such as Citeseer—benefit more substantially, and HyPIM delivers higher speedup as problem size and memory intensity increase.

b) HCNN Speedup: Compared with Hypformer, HCNN contains a substantially larger fraction of hyperbolic computation. Each stage repeatedly transforms features between Euclidean space and the Lorentz model, applies hyperbolic convolutions, and performs curvature-aware reductions. These

conversions and element-wise nonlinearities occur in every block and account for a much larger fraction of the total inference cost than in sequence models. Consequently, HCNN benefits more from HyPIM because it contains a higher proportion of hyperbolic kernels.

D. Unit Placement Analysis

We evaluate two unit-placement strategies in HyPIM: assigning one GU to each bank versus each bank group. Bank-level placement provides four times as many GUs as BG-level placement, but achieves only about 2 $\times$ higher hyperbolic-operator performance. Concurrent bank-level GUs contend for the shared TU, GB-TB path, and pCH issue slots, preventing ideal 4 $\times$ scaling. TU requests arrive in bursts as multiple banks reach transcendental stages simultaneously, creating serialization at the TU. Therefore, bank-level placement remains faster, whereas BG-level placement provides a more attractive tradeoff among performance, area, and power.

V. CONCLUSION

This work analyzes the hybrid structure of hyperbolic NNs and identifies that hyperbolic operators consist of memory-intensive reductions and transcendental primitives, along with arithmetic-intensive linear operations. We introduced HyPIM, a heterogeneous GPU-PIM architecture with GEMV units, transcendental units, and a hyperbolic-aware data layout and scheduler. Our evaluation demonstrated that HyPIM achieves up to 80 $\times$ operator speedup and 3 $\times$ end-to-end speedup over an A100 GPU.

REFERENCES

- [1] N. F. Noy and C. D. Hafner, "The state of the art in ontology design: A survey and comparative review," *AI Magazine*, vol. 18, no. 3, pp. 53–74, 1997. [Online]. Available: <https://ojs.aaai.org/aimagazine/index.php/aimagazine/article/view/1306>
- [2] R. Sarkar, "Low distortion delaunay embedding of trees in the hyperbolic plane," in *Graph Drawing*, ser. Lecture Notes in Computer Science, M. van Kreveld and B. Speckmann, Eds., vol. 7034. Springer Berlin Heidelberg, 2012, pp. 355–366.
- [3] M. Nickel and D. Kiela, "Learning continuous hierarchies in the lorentz model of hyperbolic geometry," in *Proceedings of the 35th International Conference on Machine Learning (ICML)*, 2018. [Online]. Available: <https://arxiv.org/abs/1806.03417>
- [4] O.-E. Ganea, G. Bécigneul, and T. Hofmann, "Hyperbolic neural networks," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 31, 2018. [Online]. Available: <https://arxiv.org/abs/1805.09112>
- [5] A. Bdeir, K. Schwethelm, and N. Landwehr, "Fully hyperbolic convolutional neural networks for computer vision," 2024. [Online]. Available: <https://arxiv.org/abs/2303.15919>
- [6] M. Yang, H. Verma, D. C. Zhang, J. Liu, I. King, and R. Ying, "Hypformer: Exploring efficient transformer fully in hyperbolic space," in *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '24)*. Barcelona, Spain: ACM, 2024, pp. 3770–3781.
- [7] G. Heo, S. Lee, J. Cho, H. Choi, S. Lee, H. Ham, G. Kim, D. Mahajan, and J. Park, "Neupims: Npu-pim heterogeneous acceleration for batched llm inferencing," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2024, pp. 1–16.
- [8] X. Lu, L. Hu, H. Huang, P. Jiang, and X.-H. Sun, "I/O-Aware PIM Acceleration for Long-Sequence LLM Inference with Hybrid Sparse Attention," in *2026 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2026, pp. 445–457.
- [9] Y. Gu, A. Khadem, S. Umesh, N. Liang, X. Servot, O. Mutlu, R. Iyer, and R. Das, "Pim is all you need: A cxl-enabled gpu-free system for large language model inference," in *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2025. [Online]. Available: <https://arxiv.org/abs/2502.07578>
- [10] J. Gómez-Luna, I. E. Hajj, I. Fernandez, C. Giannoula, G. F. Oliveira, and O. Mutlu, "Benchmarking a new paradigm: An experimental analysis of a real processing-in-memory architecture," 2022. [Online]. Available: <https://arxiv.org/abs/2105.03814>
- [11] J. Park, J. Choi, K. Kyung, M. J. Kim, Y. Kwon, N. S. Kim, and J. H. Ahn, "AttAcc: Unleashing the power of pim for batched transformer-based generative model inference," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '24)*, 2024, pp. 103–121.
- [12] S. Lee, S.-h. Kang, J. Lee, H. Kim, E. Lee, S. Seo, H. Yoon, S. Lee, K. Lim, H. Shin, J. Kim, S. O, A. Iyer, D. Wang, K. Sohn, and N. S. Kim, "Hardware architecture and software stack for pim based on commercial dram technology," in *Proceedings of the 48th Annual International Symposium on Computer Architecture*, ser. ISCA '21. IEEE Press, 2021, p. 43–56. [Online]. Available: <https://doi.org/10.1109/ISCA52012.2021.00013>
- [13] Y. He, H. Mao, C. Giannoula, M. Sadrosadati, J. Gómez-Luna, H. Li, X. Li, Y. Wang, and O. Mutlu, *PAPI: Exploiting Dynamic Parallelism in Large Language Model Decoding with a Processing-In-Memory-Enabled Computing System*. New York, NY, USA: Association for Computing Machinery, 2025, p. 766–782. [Online]. Available: <https://doi.org/10.1145/3676641.3716009>
- [14] J. Liu, M. Zhou, Y. Pan, C.-Y. Yang, L. Josipovic, and T. Rosing, "Optipim: Optimizing processing in-memory acceleration using integer linear programming," in *The 52nd IEEE/ACM International Symposium on Computer Architecture (ISCA)*. ACM/IEEE, 2025, pp. 1–16.
- [15] M. Zhou, W. Xu, J. Kang, and T. Rosing, "Transpim: A memory-based acceleration via software-hardware co-design for transformer," in *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2022, pp. 1071–1085.
- [16] M. Zhou, Y. Nam, P. Gangwar, W. Xu, A. Dutta, C. Wilkerson, R. Cammarota, S. Gupta, and T. Rosing, "Fhemem: A processing in-memory accelerator for fully homomorphic encryption," *IEEE Transactions on Emerging Topics in Computing*, pp. 1–16, 2025.
- [17] Y. Pan, Z. Xia, P.-K. Hsu, L. Hu, H. Kim, J. Sharda, M. Zhou, N. S. Kim, S. Yu, T. Rosing, and M. Kang, "Stratum: System-hardware co-design with tiered monolithic 3d-stackable dram for efficient moe serving," in *Proceedings of the 58th IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '25. New York, NY, USA: Association for Computing Machinery, 2025, p. 1–17. [Online]. Available: <https://doi.org/10.1145/3725843.3756043>
- [18] M. Zhou, M. Imani, S. Gupta, Y. Kim, and T. Rosing, "Gram: graph processing in a reram-based computational memory," in *IEEE Asia and South Pacific Design Automation Conference*, 2019.
- [19] L. Yan, X. Lu, X. Chen, Y. Han, and X.-H. Sun, "Pyramid: Accelerating llm inference with cross-level processing-in-memory," *IEEE Computer Architecture Letters*, vol. 24, no. 1, pp. 121–124, 2025.
- [20] I. Chami, R. Ying, C. Ré, and J. Leskovec, "Hyperbolic graph convolutional neural networks," in *Advances in Neural Information Processing Systems*, 2019.
- [21] H. Luo, Y. C. Tuğrul, F. N. Bostancı, A. Olgun, A. G. Yağlıkcı, , and O. Mutlu, "Ramulator 2.0: A Modern, Modular, and Extensible DRAM Simulator," 2023.
- [22] P. Sen, G. Namata, M. Bilgic *et al.*, "Collective classification in network data," *AI Magazine*, 2008.
- [23] K. Lang, "Newsweeder: Learning to filter netnews," in *ICML*, 1995.
- [24] O. Vinyals, C. Blundell *et al.*, "Matching networks for one shot learning," in *NeurIPS*, 2016.
- [25] J. Deng *et al.*, "Imagenet: A large-scale hierarchical image database," in *CVPR*, 2009.
- [26] A. Krizhevsky, "Learning multiple layers of features from tiny images," University of Toronto, Tech. Rep., 2009.
- [27] Z. Liu, P. Luo, X. Wang, and X. Tang, "Deep learning face attributes in the wild," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015.
- [28] T. Ajayi *et al.*, "Openroad: Toward a self-driving, open-source digital layout implementation tool," in *Proceedings of the 2020 IEEE/ACM International Conference On Computer-Aided Design (ICCAD)*, 2020.
- [29] L. T. Clark, V. Vashishtha, L. Shifren, A. Gujja, S. Sinha, B. Cline, C. Ramamurthy, and G. Yeric, "Asap7: A 7-nm finfet predictive process design kit," *Microelectronics Journal*, vol. 53, pp. 105–115, 2016. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S002626921630026X>
- [30] A. Shafaei, Y. Wang, X. Lin, and M. Pedram, "Fincacti: Architectural analysis and modeling of caches with deeply scaled finfet devices," in *2014 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 2014, pp. 290–295.
- [31] M.-J. Park, H. S. Cho, T.-S. Yun, S. Byeon, Y. J. Koo, S. Yoon, D. U. Lee, S. Choi, J. Park, J. Lee, K. Cho, J. Moon, B.-K. Yoon, Y.-J. Park, S. muk Oh, C. K. Lee, T.-K. Kim, S.-H. Lee, H.-W. Kim, Y. Ju, S.-K. Lim, S. G. Baek, K. Y. Lee, S. H. Lee, W. S. We, S. Kim, Y. Choi, S.-H. Lee, S. M. Yang, G. Lee, I.-K. Kim, Y. Jeon, J.-H. Park, J. C. Yun, C. Park, S.-Y. Kim, S. Kim, D.-Y. Lee, S.-H. Oh, T. Hwang, J. Shin, Y. Lee, H. Kim, J. Lee, Y. Hur, S. Lee, J. Jang, J. Chun, and J. Cho, "A 192-gb 12-high 896-gb/s hbm3 dram with a tsv autocalibration scheme and machine-learning-based layout optimization," in *ISSCC*, vol. 65, 2022, pp. 444–446.
- [32] Y. Ryu, S.-G. Ahn, J. H. Lee, J. Park, Y. K. Kim, H. Kim, Y. G. Song, H.-W. Cho, S. Cho, S. H. Song, H. Lee, U. Shin, J. Ahn, J.-M. Ryu, S. Lee, K.-H. Lim, J. Lee, J. H. Park, J.-S. Jeong, S. Joo, D. Cho, S. Y. Kim, M. Lee, H. Kim, M. Kim, J.-S. Kim, J. Kim, H. G. Kang, M. Lee, S.-R. Kim, Y.-C. Kwon, Y. Y. Byun, K. Lee, S. Park, J. Yoon, M.-O. Kim, K. Sohn, S. Hwang, and J. Lee, "A 16 gb 1024 gb/s hbm3 dram with source-synchronized bus design and on-die error control scheme for enhanced ras features," *IEEE Journal of Solid-State Circuits*, vol. 58, no. 4, pp. 1051–1061, 2023.
- [33] J.-H. Han, R. E. West, K. Torres-Castro, N. Swami, S. Khan, and M. Stan, "Power and thermal modeling of in-3d-memory computing," in *2021 International Symposium on Devices, Circuits and Systems (ISDCS)*, 2021, pp. 1–4.
- [34] J. H. Kim, D. Lee, G. Pekhimenko, J. Jeong, T. F. Wenisch, O. Mutlu, P. Dubey, and R. Das, "Fine-grained DRAM: Energy-efficient DRAM for portable devices," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2017, pp. 51–63.