DREAM: In-DRAM Bit-Serial PIM with Data Reuse and Efficient Mapping

PACT 2026 Submission **#34** – Confidential Draft – Do NOT Distribute!!

Anonymous Author(s)

ABSTRACT

In-DRAM Processing-In-Memory (DRAM-PIM) has emerged as a promising approach to accelerate memory-intensive workloads by mitigating data transfer overhead between DRAM and the host processor. Bit-serial DRAM-PIM (PUD) architectures further enhance efficiency by supporting runtime variable data precision, which is critical for emerging workloads such as large language model (LLM) inference. However, existing works suffer from two orthogonal limitations: *per-MAC inefficiency* arising from the lack of bit-level operand reuse, and *mapping inefficiency* arising from mapping schemes tightly coupled to a single architectural assumption. We present *DREAM*, an in-DRAM bit-serial PIM architecture that resolves both. *DREAM* introduces (i) a per-bank locality buffer that caches operand bit-rows, enabling bit-level reuse; (ii) a multi-level (bank/rank/array) broadcasting hierarchy that exposes flexible operand routing across DRAM hierarchies; and (iii) a generalized mapping framework that decouples mapping from architectural parameters. Against an equal-silicon single-GPU NVIDIA H100 system on four LLMs, *DREAM* delivers 46.3× and 6.4× geometric-mean end-to-end speedups on decode-dominated (Code Generation) and long-context (Context Understanding) scenarios, at 1.17% peripheral-area overhead, while out-performing similarly provisioned PUD and word-PE PIM baselines by geomeans of 112.8× and 5.8×.

ACM Reference Format:

Anonymous Author(s). 2026. ***DREAM***: In-DRAM Bit-Serial PIM with Data Reuse and Efficient Mapping: PACT 2026 Submission **#34** – Confidential Draft – Do NOT Distribute!!. In . ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

As large language models (LLMs) continue to grow in scale and complexity, sustained data movement between memory and compute has become the dominant performance and energy bottleneck. LLM inference comprises a compute-bound prefill phase dominated by general matrix–matrix multiplications (GEMM) and a memory-bound decode phase dominated by matrix–vector multiplications (GEMV); GEMV's low arithmetic intensity leaves even modern GPUs paired with high-bandwidth memory starved for data, bounding end-to-end throughput by memory bandwidth rather than by raw compute capability [3, 12, 60]. In response, major memory vendors—Samsung, SK Hynix, and Micron—are commercializing processing-in-memory (PIM) and processing-near-memory (PNM) architectures that move compute closer to data and exploit DRAM's high internal bandwidth [10, 23, 42, 80].

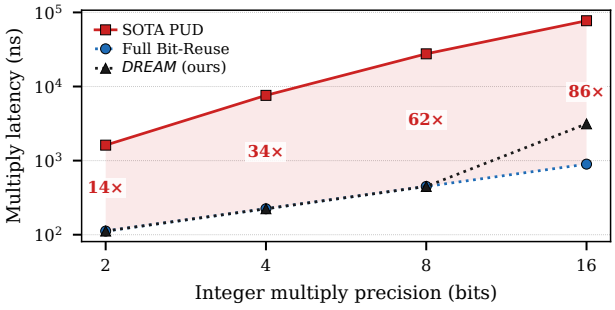


Figure 1: Integer multiplication latency vs. operand width. Existing PUD [14] pays a row activation per bit access, scaling as $O(n^2)$; bit-level operand reuse via a per-bank locality buffer reduces this to $O(n)$. Annotated multipliers in the shaded band give the per-MAC gap. *DREAM* tracks the lower bound through int8; the int16 deviation reflects an LB sized for $\leq$ int8.

Beyond bandwidth, modern DNN and LLM workloads increasingly require runtime-variable precision [29, 50, 68, 85], since their layers exhibit varying numerical sensitivity. Among precision-scalable PIM mechanisms—bit-serial, bit-sliced, and mixed-precision—bit-serial PIM is particularly appealing: it scales naturally with DRAM row parallelism and adjusts effective precision through serialization. Recent processing-using-DRAM (PUD) systems [14, 20, 21, 55] have demonstrated that DRAM arrays can execute massively parallel bit-serial logic via charge sharing within largely standard ACT/PRE command sequences, achieving high area efficiency and improved performance-per-mm^2 on memory-bound kernels.

Despite this momentum, existing PIM accelerators remain end-to-end uncompetitive with GPUs. We attribute this gap to two orthogonal limitations of prior DRAM-PIM, illustrated in Figs. 1 and 2.

Per-MAC inefficiency. PUD designs execute logic inside the physical array via charge-sharing triple-row activation, requiring an ACT/PRE pair for every bit-level operation. Because operand bits cannot be reused across successive partial products, an n-bit integer multiply scales as $O(n^2)$ and runs orders of magnitude above the analytical $O(n)$ bit-serial lower bound (Fig. 1): the gap grows from 14× at int2 to 86× at int16.

Mapping inefficiency. Prior PIM mapping schemes are tightly coupled to a single architectural assumption—hand-crafted layouts

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

Conference'17, July 2017, Washington, DC, USA

© 2026 Copyright held by the owner/author(s). Publication rights licensed to ACM.

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

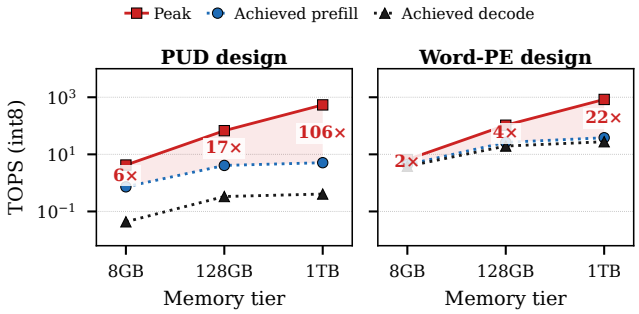


Figure 2: Achieved int8 throughput (geometric mean over fitting models) vs. peak compute across three memory tiers, for a PUD-class design [14] and a Word-PE-class design [23]. The shaded band and the annotated multiplier indicate the peak-vs-achieved gap that motivates *DREAM*

for in-array PUD logic [14, 21], fixed patterns built around a single word-PE computation model [23], or architecture-specific heuristics [55]. None expose the full mapping space, leaving the mapper unable to redistribute parallelism across the channel/rank/bank/device (CRBD) hierarchy. Fig. 2 quantifies the consequence: at the 1 TB tier, the PUD design leaves a **106**× peak-vs-achieved gap and the word-PE design leaves a **22**× gap.

To address both within a unified design, we present *DREAM*, an in-DRAM bit-serial PIM architecture comprising three mechanisms:

- **Locality buffer (LB).** A per-bank SRAM that caches operand bit-rows, enabling bit-level reuse across partial products and closing the per-MAC gap of Fig. 1.
- **Multi-level broadcasting hierarchy.** Bank-, rank-, and array-level broadcast units that route dynamic operands across the CRBD parallelism dimensions, replacing the single global bus of word-PE designs [23] and the host-driven duplication of PUD designs [20, 21].
- **Generalized mapping framework.** A formalization of the (tile, array)-mapping space that decouples mapping from architectural parameters, in contrast to prior hand-crafted [14, 21], fixed [23], and architecture-specific heuristic [55] schemes.

Against a equal-silicon single-GPU NVIDIA H100 system, *DREAM* delivers geometric-mean end-to-end speedups of **46.3**× on Code Generation (decode-dominated) and **6.4**× on Context Understanding (long-context, 8192-token input), at a peripheral-area overhead of 1.17% of DRAM.

The paper makes the following contributions:

- We identify per-MAC inefficiency and mapping inefficiency as two orthogonal limitations of prior DRAM-PIM and present *DREAM* as the first architecture to resolve both.
- We introduce a per-bank locality buffer that enables bit-level operand reuse, eliminating the per-MAC inefficiency of bit-serial PUD.
- We propose a multi-level (bank/rank/device/column) broadcasting hierarchy that, at identical peak compute, raises prefill utilization to ∼41% at the 1 TB tier.
- We develop a generalized analytical mapping framework that decouples mapping from architectural parameters and exhaustively searches the (tile, array)-mapping space.
- Against an equal-silicon single-GPU H100 system on four LLMs, *DREAM* delivers 46.3× and 6.4× geometric-mean end-to-end speedups on Code Generation and Context Understanding at 1.17% peripheral-area overhead, while outperforming similarly provisioned PUD and word-PE PIM baselines by geomeans of 112.8× and 5.8×.

The remainder of this paper is organized as follows. Section 2 reviews DRAM organization, bit-serial computation, and matrix-multiplication tiling. Section 3 describes the proposed *DREAM* architecture and microarchitecture. Section 4 presents the mapping and scheduling framework. Sections 5 and 6 present the evaluation methodology and experimental results. Sections 7, 8, and 9 discuss broader implications, related work, and conclusions.

2 BACKGROUND

2.1 DRAM Organization

DRAM is hierarchically organized into *channels*, *ranks*, *devices*, *banks*, and *sub-arrays* [21, 33, 55, 66]; we abbreviate this hierarchy as *CRBD* and the per-bank sub-array dimension as *A* throughout. Within a sub-array, a row activation latches the addressed wordline into the row buffer via the sense amplifiers, after which column accesses read or write the buffered row.

2.2 Bit-Serial Computation

Bit-serial computation processes one bit of each operand per cycle and, when implemented as a SIMD scheme across many lanes, matches or exceeds bit-parallel throughput while making operand precision a runtime parameter: an n-bit operation completes in $O(n)$ cycles. In-DRAM bit-serial requires a vertically transposed layout in which bits of equal significance share a DRAM row across sub-array columns [14, 21, 52, 55]; static operands such as model weights are transposed offline. We refer the reader to prior work [4, 21, 55] for the detailed arithmetic.

2.3 Buffered DIMMs

Server DRAM is delivered on *load-reduced* DIMMs (LRDIMMs) [19, 67], which place a *Data Buffer* (DB) chip per rank to decouple host-side DQ from the rank-side stub, enabling more ranks per channel than unbuffered DIMMs. Each DB normally performs a 1-to-1 redrive between one host-side and one rank-side DQ slice; *DREAM*'s rank- and device-level broadcasters (Section 3.5) extend it with a multi-rank-target write mode.

3 *DREAM* ARCHITECTURE

3.1 System Overview

DREAM augments conventional DRAM with three classes of additional units and extends the host–DRAM command interface to enable reuse-aware in-DRAM processing. As illustrated in Fig. 3, *DREAM* integrates: (1) per-bank compute units—bit-serial processing elements (PEs), a locality buffer, and a popcount reduction unit; (2) a per-device finite-state machine (FSM) shared by all banks for control; and (3) broadcast units operating at the bank, rank, device

Table 1: Extended PIM Commands and Their Instruction Encodings.

Instruction	Opcode Field	Operand Fields	Control Field	Description
pim_enable	000010	–	–	Enables the PIM operating mode through a Mode Register Set (MRS) write
pim_disable	000011	–	–	Disables PIM mode and restores normal DRAM command decoding
broadcast_enable	000000	–	bank_bc, col_bc	Enables broadcast write mode of bank or column
broadcast_disable	000001	–	–	Disables broadcast mode
pim_add	010000	R_{dst}, R_{src1}, R_{src2}	prec[3:0]	Performs bit-serial addition of operands R_{src1} and R_{src2}, storing the result in R_{dst}.
pim_mul	010001	R_{dst}, R_{src1}, R_{src2}	prec[3:0]	Performs bit-serial multiplication between R_{src1} and R_{src2}, storing product in R_{dst}.
pim_mul_red	010010	R_{dst}, R_{src1}, R_{src2}	prec[3:0]	Performs bit-serial multiplication followed by column-wise popcount reduction.
pim_add_parallel	010011	R_{dst}, R_{src1}, R_{src2}	-	Performs bit-parallel addition using the adder inside the popcount reduction unit.

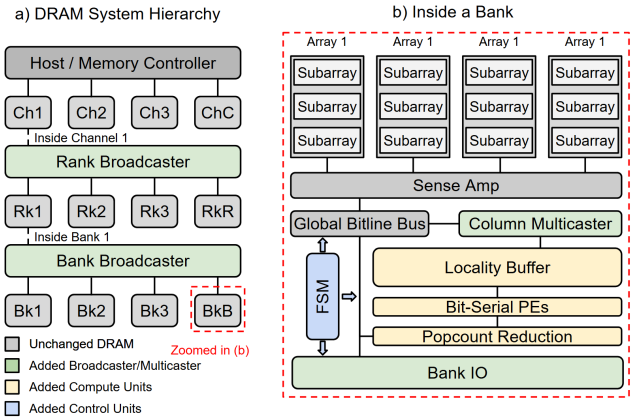
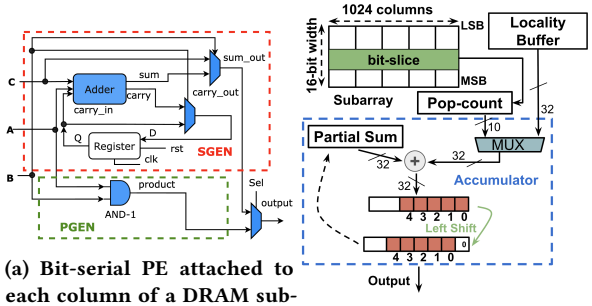


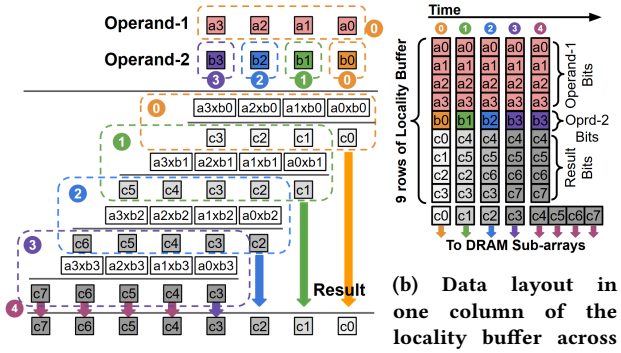
Figure 3: *DREAM* system overview. Added peripheral units are highlighted in yellow.



(a) Bit-serial PE attached to each column of a DRAM sub-array. The product is generated by *PGEN* and the partial sum by *SGEN*.

(b) Popcount reduction unit for 16-bit partial sums across 1024 sub-arrays.

Figure 4: Peripheral units added by the *DREAM* architecture.



(a) Multiplication operation and execution steps.

(b) Data layout in one column of the locality buffer across steps; identical for all columns.

Figure 5: Example of int4 bit-serial multiplication.

levels and multicast units at the array level. The PIM command interface, summarized in Table 1, is encoded using vendor-reserved or unused command/address slots in the DDR5 protocol. The remainder of this section describes each architectural component.

3.2 Bit-Serial PE

Each bank contains a bit-serial PE array that performs all in-DRAM arithmetic (Fig. 4a). Every PE exposes two functional primitives: a 1-bit full adder with carry (*SGEN*) and a 1-bit multiplier (*PGEN*). These primitives compose into arbitrary-precision integer addition and multiplication by consuming operand bits over successive cycles, so an n-bit multiply-accumulate completes in $O(n)$ cycles. Operand precision is consequently a runtime parameter rather than a fixed datapath width; the host invokes these operations through the `pim_add`, `pim_mul`, and `pim_mul_red` commands of Table 1.

The PE array sources its operands from the per-bank locality buffer (Section 3.3) rather than directly from the DRAM array, and a single PE cycle advances one bit-plane across the entire buffer row in parallel. The array thus presents as a SIMD vector whose width equals the locality-buffer row—1024 lanes in our default configuration.

3.3 Locality Buffer and Bit-Serial Multiplication

DREAM introduces a per-bank SRAM *locality buffer* (LB) that caches operand bit-rows adjacent to the PE array and supplies all bit-serial operands. The LB transforms bit-serial multiplication from an $O(n^2)$ DRAM-access pattern—in which each operand bit is re-fetched from the array once per partial product—into an $O(n)$ pattern in which each operand bit is read from the array exactly once.

Fig. 5 illustrates the resulting dataflow. One operand is held *stationary* in the LB: its n bit-rows are loaded once and reused for the entire multiplication. The other operand is *streamed*, one bit-row per cycle, and each freshly loaded row gates a conditional add of the stationary operand into a running result, also buffered in the LB. Because the stationary operand is never re-read and each streamed bit is read exactly once, an n-bit multiply incurs only $O(n)$ array-to-LB fills.

The PE cycle is sized to match the LB-row fill time t_{LB}, so the bit-serial update consuming the previous row completes in the same interval used to load the next. Bit-serial computation is therefore pipelined behind the streaming LB load, and multiplication becomes

load-bound: latency is $O(n \cdot t_{LB})$, set by the number of LB fills rather than by PE logic depth. Sustaining this pipeline requires the LB to concurrently hold the stationary operand, the current streamed row, and the accumulating result—$2n + 1$ rows for n-bit precision. We provision 17 rows per bank, sufficient for full reuse up to int8 multiplication.

A second, coarser level of reuse arises when the stationary operand participates in multiple independent multiplications against different streamed operands while remaining resident in the LB. Whenever the workload mapping (Section 4) exposes such reuse—for example, when an activation row is multiplied by many weight rows in a GEMV tile during LLM decoding—the n-row cost of loading the stationary operand is amortized over all reusing multiplies, and the per-multiply latency asymptotes to the streamed-operand fill cost.

3.4 Popcount Reduction

When the reduction axis (K) of a matrix multiplication is mapped across the PE array's SIMD lanes, the partial products produced by different lanes must be summed to a single scalar. *DREAM* performs this reduction using a dedicated per-bank *popcount reduction unit* (Fig. 4b) co-located with the PE array and the locality buffer. A conventional word-level adder tree is unsuitable for the vertical bit-serial layout, since it would first have to reassemble full-width operands across cycles. The popcount unit instead reduces directly in the bit-serial domain, processing one bit-slice per cycle through a compact, log-depth counting tree on 1-bit inputs.

For each incoming bit-slice of significance i, the unit contributes $\text{popcount}(\text{bit-slice}_i) \cdot 2^i$ to a running sum. To minimize area, the physical popcount tree is 128 bits wide—8× narrower than the 1024-lane SIMD—and operates at the 1.25 ns PE clock, time-multiplexing across 8 cycles to process a full 1024-wide bit-slice. This sizes per-slice throughput to one LB-row fill (10 ns), so the unit accepts a new bit-slice per fill without back-pressure. Reducing an n-bit partial sum across the SIMD lanes therefore completes in n LB-row intervals, with reduction latency determined entirely by the partial-product bit-width.

The `pim_mul_red` command fuses multiplication with column-wise reduction, feeding each partial-product bit-plane into the popcount unit as it is produced. The reduced scalar is written back to the DRAM array in horizontal (word-aligned) layout, avoiding the row activations a vertical writeback would require. The unit's accumulator additionally serves as the wide bit-parallel adder invoked by `pim_add_parallel` to merge reduced scalars across temporal tiles.

3.5 Broadcasting Units

DREAM supports peer-to-peer broadcast across any subset of the four parallelism dimensions $\{C, R, B, D\}$ via a hierarchical tree of store-and-forward turn-around points, each placed immediately above the level it fans out. A bank-level (B) broadcast propagates across banks within a device over an extended on-die global bitline bus—a generalization of the intra-bank GBB to device scope. A rank- or device-level (R, D) broadcast uses an LRDIMM-class data-buffer (DB) chip on the DIMM, which captures the source rank's column-read output from the shared DQ pins and re-drives DQ as a broadcast write to all peer ranks (replicating the per-device 16 b slice rather than striping it for D-level fanout). A channel-level (C) broadcast uses the memory controller's cross-channel interconnect to forward captured data to the DBs of peer channels. Composite broadcast masks (e.g., $\{C, B\}$) chain these turn-around points: data flows from the source bank up through its DB to the memory controller, across the controller fabric to each peer channel, down to the destination device I/O, and onto each device's extended GBB to reach all listening banks. Every broadcast terminates in a destination bank's locality buffer (SRAM)—never in the DRAM array—and therefore consumes no t_{RCD}, t_{RP}, or t_{RC} array cycles, proceeding concurrently with column accesses on the same bank. Operands consequently require no physical duplication in DRAM capacity; duplication exists only transiently in locality buffers as broadcasts traverse the tree during compute.

Broadcasts stream at the granularity of individual locality-buffer rows. Each bank's locality buffer is organized as a small double buffer, allowing one row to absorb an incoming broadcast while the PE array consumes the previous row bit-serially. Because a row's broadcast transit is substantially shorter than the bit-serial compute time that consumes it, broadcasts pipeline cleanly under compute, and the PE array does not stall on the broadcast fabric in steady state. The only overhead that remains visible is the one-time latency of filling the broadcast tree at the start of a compute phase; once the pipeline is primed, broadcasting sustains the PE array at full bit-serial throughput.

4 WORKLOAD MAPPING

DREAM's mapping framework targets arbitrary-shaped matrix–matrix multiplications (matmuls) on arbitrary *DREAM* hardware configurations. It decomposes the mapping problem into two levels: a *Tile Mapping* that distributes the matmul across the DRAM parallelism hierarchy, and an *Array Mapping* that fixes the data layout and compute scheme inside each bank's compute substrate.

Physical and logical arrays. A physical DRAM sub-array is typically far wider than the compute substrate attached to it: our default configuration pairs a 1024-lane PE array with sub-arrays of tens of thousands of columns. The framework therefore operates on a *logical array* abstraction rather than on raw sub-arrays. A logical array is a vertical slice of a physical sub-array whose column count matches the PE/locality-buffer width. In *DREAM*, each bank contains a single compute substrate—one set of PEs, locality buffers, and popcount units—and consequently exposes exactly one logical array per bank. The remaining column capacity of the physical sub-array is reached by temporal tiling over this array. The framework itself is more general: architectures that provision compute at sub-array granularity can expose multiple logical arrays per bank, surfacing sub-array-level parallelism as an additional spatial dimension. A direct projection from logical arrays to physical column positions yields the final DRAM layout, keeping mapping decisions independent of the underlying sub-array width.

Two mapping levels. Tile Mapping assigns each matmul dimension (M, K, N) to one or more parallelism levels: C (channel), R (rank), B (bank), D (device), and A (logical array within a bank). A dimension mapped to multiple levels is hierarchically tiled across

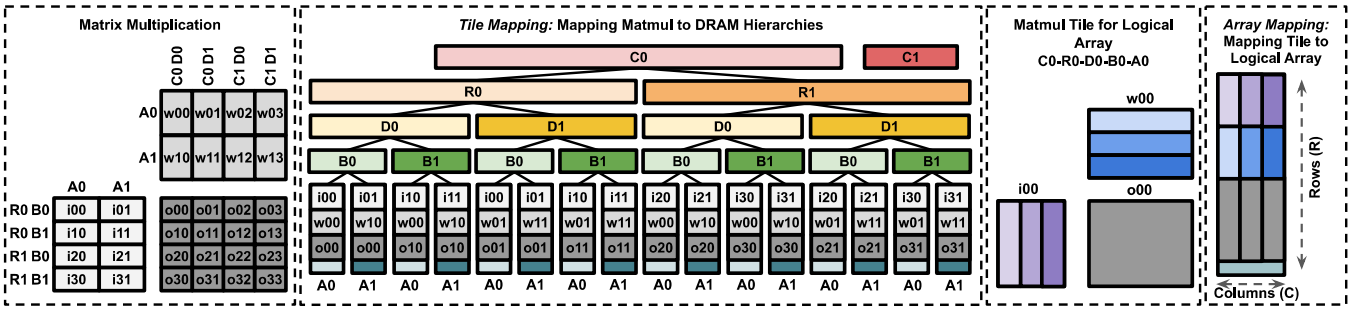


Figure 6: Example of Tile Mapping {*M: RB, N: CD, K: A*} and Array Mapping {*R: MN, C: K*} for a matmul kernel.

them; a dimension mapped to none is broadcast across all slots at those levels. Array Mapping then assigns matmul dimensions to the row (R) and column (C) axes of a logical array, fixing the in-array data layout and the contraction axis along which bit-serial SIMD operates. Fig. 6 illustrates the running example used throughout this section: Tile Mapping {*M: RB, N: CD, K: A*} with Array Mapping {*R: MN, C: K*}.

4.1 Tile Mapping

Under {*M: RB, N: CD, K: A*} (Fig. 6, two channels, ranks, banks, devices, and logical arrays per bank; channel-1 omitted), M is tiled across ranks and banks, N across channels and devices, and K across logical arrays within each bank. The per-array sub-tile and required broadcasts follow directly from the (C, R, B, D, A) indices: each input tile is replicated across the parallelism dimensions to which its dimensions are not mapped, serviced by the broadcast fabric (Section 3.5) rather than physical duplication. Cross-array reduction is implied wherever K is distributed across a parallelism dimension.

4.2 Array Mapping

Array Mapping {$R: M_t N_t$, $C: K_t$} (Fig. 6, right) lays each sub-tile (M_t, K_t, N_t) across the logical array so that each column holds one slice of input tile i_{00}, one slice of weight tile w_{00}, and the partial sum of o_{00}; the K_t contraction runs along the column axis in SIMD. The column-wise multiplication and reduction are fused into a single `pim_mul_red` (Section 3.4).

4.3 Temporal Tiling

Tile Mapping and Array Mapping together determine the *spatial* placement of a matmul on the DRAM. When the per-array sub-tile exceeds the capacity of a single logical array, a final *temporal tiling* pass divides it into smaller tiles iterated in time on the same array. Under the running Array Mapping {*R: MN, C: K*}, a sub-tile fits natively when $K_t \le$ #Cols and $M_t \cdot N_t \le$ #Rows of the logical array; otherwise the number of temporal iterations required to cover the full matmul is $\frac{M}{\sqrt{\#\text{Rows}}} \cdot \frac{N}{\sqrt{\#\text{Rows}}} \cdot \frac{K}{\#\text{Cols}}$, where rows and cols refer to the logical array. Temporal tiling enables *DREAM* to handle matmuls whose footprint exceeds the on-bank compute substrate without altering the spatial mapping decisions above it.

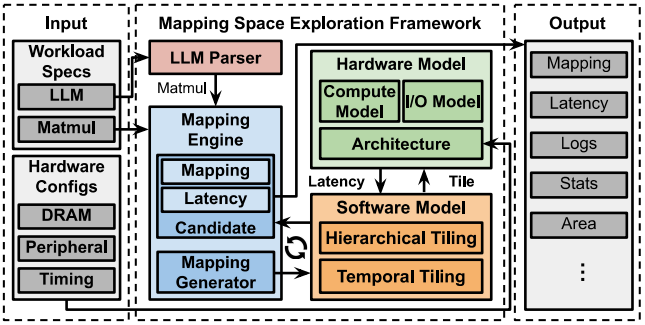


Figure 7: *DREAM* mapping framework.

4.4 Mapping Framework

Fig. 7 illustrates the *DREAM* mapping space exploration framework. The framework takes as input the workload specification (LLM hyperparameters or standalone matmul shapes) and the hardware configuration (DRAM organization, peripheral-unit configuration, and timing parameters). For each workload–hardware pair, the framework produces an optimized mapping along with the corresponding end-to-end latency and auxiliary logs, statistics, and area reports.

The framework comprises an LLM parser, a mapping engine, a software model, and a hardware model. The LLM parser decomposes an input LLM into a sequence of matmul kernels and forwards them to the mapping engine. The mapping engine maintains the current best candidate mapping and its latency. A mapping generator inside the engine enumerates the mapping space, instantiates each candidate, evaluates it via the software and hardware models, and updates the candidate when a lower-latency mapping is found.

The software model applies the tile, array, and temporal mappings according to a given strategy and issues per-tile compute and data-movement requests to the hardware model. For each tile, the hardware model returns the computation and I/O latencies; the software model accumulates these across temporal tiles. The total kernel latency is the sum of all tile compute and I/O latencies and is returned to the mapping engine as the objective value of the candidate.

The hardware model comprises an architectural description, a compute model, and an I/O model. The architecture is parameterized by the input hardware configuration. Given a tile and its

mapping, the compute model derives per-array PIM latency by summing the latencies of the PIM instructions executed on the locality buffers, PEs, and reduction units. The I/O model estimates the latency of host–DRAM interactions across DRAM hierarchies during input broadcasting and output reduction, based on the inferred data layout, the I/O traffic volume, the effective bandwidth, and the broadcast-unit configuration. Together, these models enable *DREAM* to explore the full mapping space and identify latency-optimal mappings for each kernel.

4.5 KV Caching

DREAM keeps the KV cache resident in the DRAM sub-arrays that execute attention, eliminating the per-token K/V streaming that bandwidth-bounds decode on accelerator-centric systems. The $Q \cdot K$ and softmax$(A) \cdot V$ contractions run bit-serially on the bitlines storing the cache; only the freshly generated query crosses the memory–compute boundary, and the column-popcount tree reduces partial products in place. The decode mapping pins each attention head to a distinct (channel, rank) slot, exposing full head-parallelism without tensor-parallel replication, and per-token growth reduces to a local row append within the already-allocated bank. Combined with bit-serial precision agility—int4/int2 caches yield proportional latency and footprint reductions—this mapping turns decode-time attention into a compute-bound, head-parallel kernel that scales with DRAM internal parallelism rather than off-chip bandwidth.

5 METHODOLOGY

5.1 Simulation Infrastructure

We extend *LLMCompass* [86] with the analytical models required to evaluate *DREAM*, the PUD-class baseline, and the Word-PE-class baseline within a single framework. Following prior analytical PIM modeling work [32, 49], we hierarchically model channels, ranks, banks, devices, sub-arrays, and the bit-serial compute substrate, together with the broadcast units, locality buffer, and popcount-reduction logic added by *DREAM*. DRAM timing parameters are validated against Ramulator [40] and JEDEC DDR5 datasheets. The peripheral logic is implemented in Verilog [76] and synthesized with Synopsys Design Compiler [5] on the FreePDK 45 nm library [1, 72] to obtain timing parameters and gate-level area.

5.2 Architectures Evaluated

We evaluate three PIM architectures and one GPU baseline, all configured for single-device inference. The **GPU baseline** is an equal-silicon, single NVIDIA H100 SXM5 system (80 GB HBM3, 512 GB host DDR memory), modeled with LLMCompass's tensor-core and HBM3 parameters [54]. We assume that model weights and the KV cache are accessed at full HBM bandwidth, regardless of HBM capacity; this isolates GPU compute and HBM bandwidth as the variables of interest and gives an upper-bound GPU baseline against which *DREAM*'s speedups are measured. The **PUD-class** baseline is instantiated against Proteus [14]: TRA-based bit-serial compute in the physical array with no locality buffer, no popcount tree, and host-driven operand duplication. The **Word-PE-class** baseline is instantiated against Newton/AiM [23]: one int8 multiply-accumulate unit per bank, served by a single global broadcast bus with a fixed mapping. ***DREAM*** provides 1024 logical bit-serial PEs per bank operating at 800 MHz, a per-bank locality buffer of 17 1024-bit rows, a column popcount-reduction tree, and the multi-level broadcast units described in Section 3.

5.3 Memory-Tier Sweep

We evaluate three memory tiers—**8 GB**, **128 GB**, and **1 TB**—obtained by sweeping channels (1–8), ranks (1–16, supported by LRDIMM), banks (32, JEDEC DDR5 mandate), and per-die capacity (8 Gb / 16 Gb DDR5). For each (architecture, tier, workload) cell we report the lowest-latency configuration. *DREAM* additionally fixes *one* configuration per tier, selected as the geometric-mean optimum across decode workloads, reflecting our proposal as a single deployable architecture. The Word-PE and PUD baselines retain per-workload best configurations, a methodological choice that favors the baselines.

5.4 Workloads

We evaluate four LLMs that span dense and grouped-query attention: GPT-3 (175 B and 6.7 B; multi-head) [6] and Llama-3.1 (70 B and 8 B; grouped-query, $n_{\text{kv}} = 8$) [74]. All workloads use `int8` and batch size 1, the standard operating point for single-user edge inference. *Standalone phases:* prefill at 1024 input tokens; decode at 2048 output tokens after a 1024-token prompt. *End-to-end scenarios*, following [26, 44]: *Code Generation* (1024 input / 4096 output, modeling on-device coding assistants) and *Context Understanding* (8192 input / 256 output, modeling local long-document analysis or retrieval-augmented generation).

5.5 Area Estimation Methodology

PIM area is computed as *DRAM area + peripheral area*. We estimate DRAM area from Micron's reported 66 mm^2 per 16 Gb DDR5 die [75], assuming constant per-bit density; the locality-buffer SRAM uses TSMC 45 nm SRAM bit-density [83]. Peripheral logic (bit-serial PEs, popcount tree, broadcast units, control) is synthesized in FreePDK 45 nm and scaled to 14 nm—one node behind contemporary DDR5 fabrication, as DRAM peripheral circuits typically use older nodes and fewer metal layers [31, 58, 69, 71]. Post-synthesis area is corrected for placement utilization, buffer growth, and routing-layer count using prior analytical projections [35, 36, 70]. The Word-PE baseline charges only int8 multiplier and adder area per word-PE (no locality buffer or popcount logic); the PUD baseline approximates peripheral overhead at 1% of DRAM area, consistent with reported in-array TRA designs [14, 66]. The single-GPU system comprises an H100, its 80 GB HBM, and 512 GB of host DDR5 provisioned to hold models exceeding HBM capacity, giving the GPU system and *DREAM*-1 TB comparable total silicon footprints. For performance-per-area, we charge the GPU only its die and HBM area; the host DDR5 is not included. All area figures are reported at 14 nm.

6 RESULTS

In this section, we evaluate *DREAM* along four metrics: *end-to-end throughput*, *prefill throughput*, *decode throughput*, and *performance per mm*^{2}. We compare *DREAM* against an NVIDIA H100 GPU and two PIM baselines—a PUD-class baseline (Proteus) and a Word-PE-class baseline (Newton/AiM)—across three memory tiers: 8 GB,

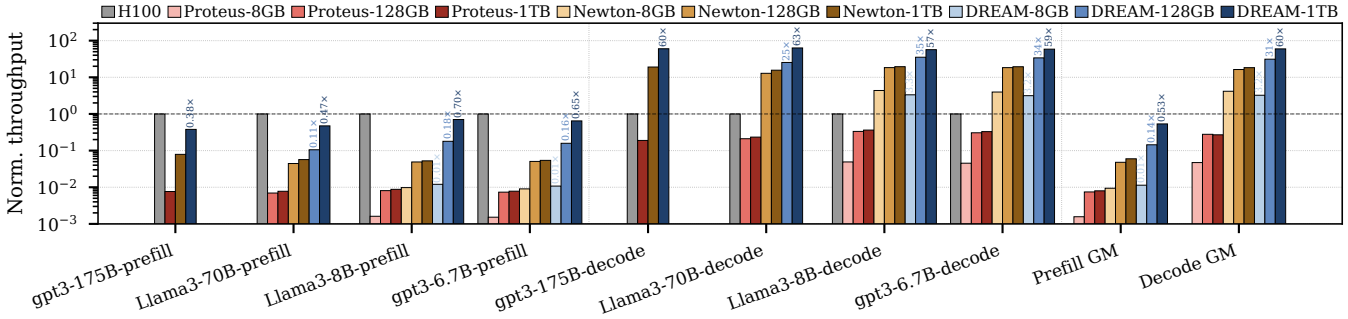


Figure 8: Latency speedup over H100 (log scale) for the PUD baseline, the Word-PE baseline, and *DREAM* at three memory tiers (8 GB, 128 GB, 1 TB), across four prefill and four decode workloads. Per-tier configurations are selected by best-of-sweep per (architecture, capacity cap, workload). Bars are omitted where the model's int8 weights exceed the tier capacity (175 B and 70 B at 8 GB; 175 B at 128 GB), and the corresponding cells are excluded from the per-tier Prefill GM and Decode GM. Annotated values denote *DREAM* ratios.

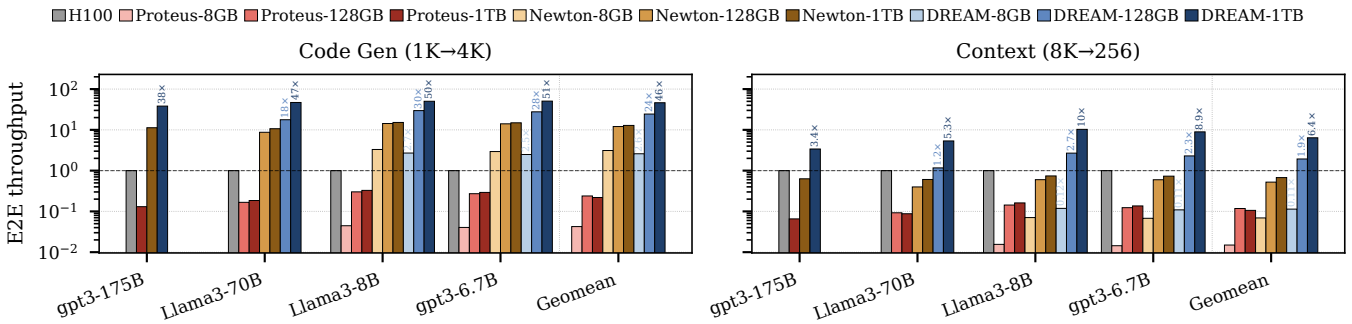


Figure 9: End-to-end throughput normalized to H100 on the *Code Generation* (1024 input / 4096 output tokens) and *Context Understanding* (8192 input / 256 output tokens) scenarios. Bars and per-tier geomeans omit (model, tier) pairs where the model's int8 weights exceed the tier capacity, as in Fig. 8.

128 GB, and 1 TB (Section 6.1). We then report area and performance per unit area (Section 6.2), conduct an architecture ablation study (Section 6.3), and present a sensitivity analysis (Section 6.4).

6.1 Performance

Fig. 8 reports prefill and decode latency speedups over H100 across three memory tiers (8 GB, 128 GB, 1 TB) for the PUD baseline, the Word-PE baseline, and *DREAM*.

Prefill. *DREAM*-1 TB delivers a geometric-mean prefill speedup of **0.53**× H100 (peak 0.7× on Llama-3.1-8B), sustaining **81**%, **76**%, and **41**% of peak compute across the 8 GB, 128 GB, and 1 TB tiers (Fig. 10). At the matched 1 TB tier this translates to **8.9**× above the Word-PE baseline (0.06× H100) and **67**× above the PUD baseline (0.008× H100).

Why DREAM trails H100. Prefill is compute-bound, so the achievable speedup is upper-bounded by raw int8 throughput. The H100 delivers 1979 Tops int8 (dense, no structural sparsity) [54], while *DREAM*-1 TB's peak is only 838.9 Tops (0.42× of H100); the smaller peak reflects the trade-off of placing compute on commodity DRAM rather than a separate compute die. The 0.53× achieved ratio is essentially compute-ceiling-bound—*DREAM* converts most of its peak into achieved throughput.

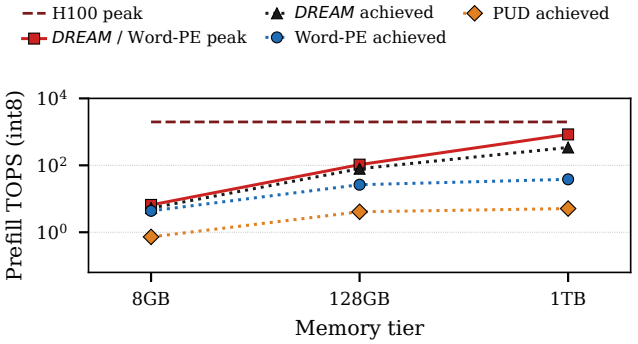


Figure 10: Achieved prefill TOPS (geometric mean over fitting models) vs. peak compute across three memory tiers, with the H100 int8 peak (1979 Tops) drawn as a horizontal reference. *DREAM* and Word-PE share an identical peak; PUD's peak is 0.64× that value (annotated in Fig. 2).

Why DREAM beats Word-PE. *DREAM* and the Word-PE baseline share an identical peak at matched configurations (6.6, 104.9, and 838.9 Tops at 8/128/1024 GB), yet *DREAM* sustains higher peak utilization at every tier (81/76/41% vs. 67/25/5%), and the gap widens

with system size. The disparity is mapping efficiency: *DREAM*'s multi-level broadcasting hierarchy (bank, rank, and array) lets the mapper flexibly distribute operands across the CRBD parallelism dimensions, whereas the Word-PE baseline's single global broadcast bus admits only one operand-routing pattern. Each added parallelism dimension therefore demands more routing flexibility to keep PEs fed—supplied by *DREAM*'s broadcasters but not by the Word-PE bus.

Why DREAM beats PUD. PUD exposes 256× more SIMD lanes than *DREAM*/Word-PE (16K columns per bank-device), but each TRA-based int8 multiply takes approximately 345× slower per operation. PUD's peak is therefore only 0.64× of *DREAM*/Word-PE (4.2, 67.5, and 540 Tops), and its wide flat SIMD is difficult to fully utilize on LLM tile dimensions, holding peak utilization under 11% at every tier. The combined per-MAC and utilization gaps yield the 67× *DREAM* advantage at 1 TB.

Decode. Decode is memory-bound: KV cache and weights are resident in the compute substrate, so aggregate internal bandwidth—rather than peak Tops—governs throughput. *DREAM*-1 TB attains a **59.6**× geometric-mean speedup (peak 63.2× on Llama-3.1-70B), 3.3× above Word-PE-1 TB (18.3×) and 220× above PUD-1 TB (0.27×); the gaps over Word-PE and PUD have the same mapping-efficiency and per-MAC-efficiency origins as in prefill.

End-to-end. For Code Generation (4096-token decode), decode overwhelmingly dominates wall-clock time and end-to-end throughput tracks decode: *DREAM* delivers a **46.3**× geometric-mean speedup over H100 (Fig. 9). For Context Understanding (8192-token input, 256-token output), the input is large in token count but, at batch 1, the GPU's autoregressive decode of 256 tokens still accounts for the majority of wall-clock time because each output token is bandwidth-bound and sequential. The prefill component is nevertheless large enough to pull prior PIM designs below H100: only *DREAM* sustains prefill performance approaching H100's, allowing it to retain the decode advantage end-to-end, and it is the only PIM design to exceed H100 (**6.4**× geometric mean) on this scenario. At the matched 1 TB tier, *DREAM* out-performs the PUD baseline by 210.9× on Code Generation and 60.3× on Context Understanding (geomean **112.8**×), and the Word-PE baseline by 3.6× and 9.5× (geomean **5.8**×).

6.2 Performance per Area

Peripheral area. A 1 TB *DREAM* system adds only **1.17**% peripheral logic on top of the underlying DRAM (396 mm^2 over 33,792 mm^2). Bit-serial PEs dominate the peripheral budget at **90.1**%, locality-buffer SRAM accounts for 6.8%, and the column popcount tree for 3.0%; broadcasters together contribute under 0.2%, since they reuse the existing DDR5 DQ wiring rather than introducing a new on-die network.

Decode performance per area. *DREAM* attains geometric-mean decode performance per area of **174.3**×, **105.0**×, and **25.1**× at the 8 GB, 128 GB, and 1 TB tiers (averaged over the models that fit at each tier). At 8 GB, the Word-PE baseline still exceeds *DREAM* (225.1× vs. 174.3×), but *DREAM* scales substantially better with capacity: at 128 GB and 1 TB, *DREAM* delivers 1.9× and 3.2× higher decode performance per area than the Word-PE baseline. The PUD baseline tops out at 2.54× H100 at 8 GB and falls to 0.11× at 1 TB.

Prefill performance per area. No PIM system exceeds H100 on prefill performance per area (*DREAM*-8 GB attains 0.61× at best, in geometric mean), consistent with the prefill compute ceilings discussed in Section 6.1. *DREAM* nevertheless degrades the most slowly with capacity: 0.61× → 0.49× → 0.22× from 8 GB to 1 TB—a 2.7× drop across a 128× silicon increase—compared with the Word-PE baseline's 0.51× → 0.16× → 0.03× (20× drop) and the PUD baseline's still steeper decline. *DREAM*'s prefill efficiency therefore scales closest to ideal as the system grows, again attributable to the flexible mapping enabled by the multi-level broadcasters.

6.3 Architecture Ablation Study

Fig. 12 reports throughput normalized to full *DREAM*-1 TB for four single-removal variants—disabling popcount reduction (PR), broadcasting units (BU), or locality buffers (LB) individually—together with a fifth variant in which all three are disabled simultaneously. The slowdown values reported below are the inverse of the figure's normalized throughput.

Locality buffer asymmetric to prefill. Removing LB slows prefill by 7.3× in geometric mean (range 6.5–8.2×) and decode by 3.6× (2.8–5.4×). The asymmetry mirrors the compute-bound profile of prefill: without LB, every bit-serial reuse falls back to the DRAM cell array, and the resulting ACT/PRE traffic is more costly in a phase that would otherwise be compute-dense.

Broadcasting unit asymmetric to decode. The opposite profile arises for BU: prefill slows by only 1.6× (1.2–1.9×), while decode slows by 6.3× (5.6–7.2×). Decode kernels read a single activation row per token, so the kernel-input duplication that BU avoids accounts for a much larger fraction of decode's I/O budget than of prefill's.

Popcount reduction symmetric. Removing PR slows prefill by 3.2× (2.6–3.7×) and decode by 4.5× (3.2–6.5×): partial sums must be exported off-array and reduced at the host in both phases, charging external bandwidth that the in-array popcount tree would otherwise hide.

All three together. Disabling PR, BU, and LB simultaneously yields 18.6× prefill and 23.6× decode slowdown, approximately equal to the product of the individual costs. The three structures address largely orthogonal bottlenecks, and removing any one inflicts disproportionate damage on either prefill or decode.

6.4 Sensitivity Study

6.4.1 ***Precision sensitivity.*** Fig. 13 sweeps operand precision from `int1` to `int16`, with each curve normalized to its system's `int8` baseline. *DREAM* achieves *exactly* linear bit-width scaling on every workload: **8.00**× at `int1`, **4.00**× at `int2`, and **2.00**× at `int4` (no spread across the eight workloads; per-workload speedups equal the geometric mean). This matches the ideal predicted by bit-serial reuse and confirms that *DREAM* introduces no fixed precision-dependent overhead in the `int1`–`int8` range. *DREAM*'s `int16` latency, however, is 27× slower than `int8` for prefill and 12× slower for decode—a discontinuity that breaks the linear regime, since 16-bit operands no longer fit within the LB-row layout on which the bit-serial pipeline depends. The PUD baseline exhibits non-ideal sub-bit-width scaling at low precisions (13.0×, 5.4×, and 2.3× for `int1`, `int2`, and `int4` respectively): its compute time is dominated

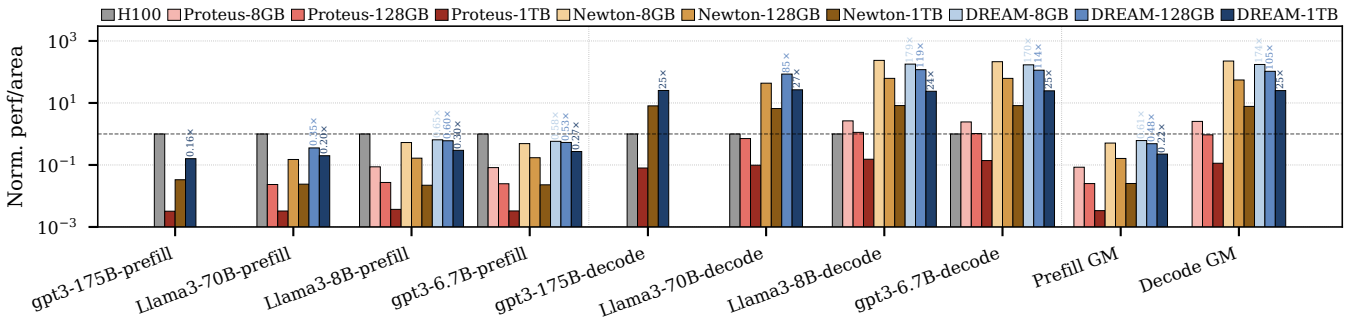


Figure 11: Performance per unit area normalized to H100. The H100 area is the sum of die area (7,994 mm^2) and a flattened 80 GB HBM (6,381 mm^2); the 512 GB of host DDR5 provisioned in the GPU edge system (Section 5.5) is not charged here. The PIM area is DRAM area plus peripheral overhead, scaled up to the tier cap when the winning sweep configuration underfills the tier (so the deployed silicon is charged). Bars and per-tier geomeans omit (model, tier) pairs where the model's int8 weights exceed the tier capacity, as in Fig. 8. All areas are reported at 14 nm. Annotated values denote *DREAM* ratios.

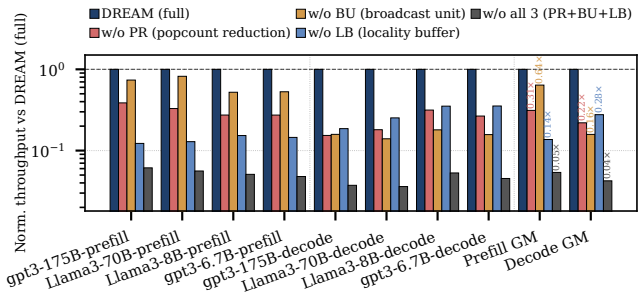


Figure 12: *DREAM* ablation study: latency normalized to the full design (locality buffer LB, popcount reduction PR, and broadcasting unit BU all enabled), with the three structures progressively removed.

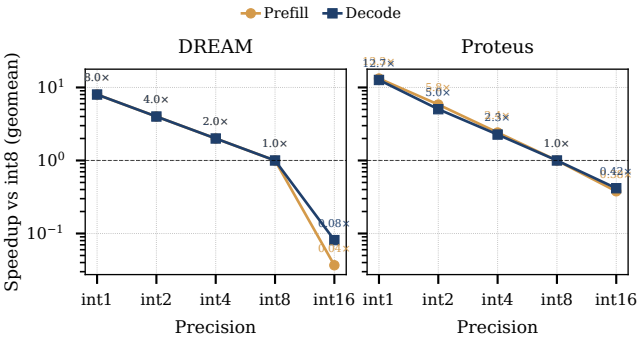


Figure 13: *DREAM* and Proteus latency versus operand precision (`int1`–`int16`), normalized to each system's `int8` baseline.

by SIMDRAM-style mul latency, but fixed I/O overheads dilute the speedup at low precision.

6.4.2 ***Mapping sensitivity.*** Fig. 14 reports a mapping sweep over a single $1024 \times 12288 \times 12288$ GEMM, comprising 486 (array × tile) mapping cells. The dominant lever is the array mapping: placing the contraction dimension K on the SIMD column axis (C=K) is 30× faster than the worst array mapping (R=MK, C=N), and the gap

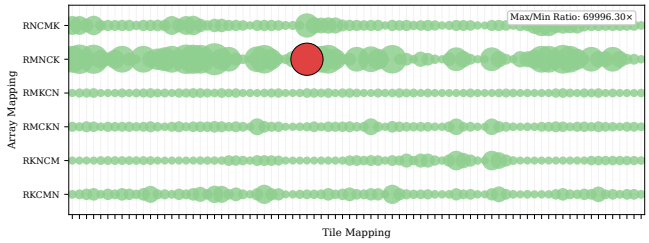


Figure 14: Mapping-strategy sensitivity for a $1024 \times 12288 \times 12288$ GEMM on *DREAM*-1TB. Each cell shows the best latency among configurations sharing that (array, tile) mapping; bubble size is proportional to $1/$latency, and the global winner is highlighted in red.

between C=K and the next-best array mapping is already 1.9×. A two-rule heuristic—fix C=K, then assign M and N to the CRBD parallelism dimensions in proportion to each dimension's parallelism budget—therefore captures most of the available speedup without search. The residual 1.9× comes from the choice of tile mapping *within* the C=K family: no single tile mapping is consistently optimal across workload shapes, so a heuristic alone leaves this gap on the table. The framework's contribution is twofold: it surfaces C=K as a workload-independent design rule that any mapper can adopt, and exhaustive search over the much smaller residual space closes the remaining gap.

7 DISCUSSION

Mapping Search Cost. Our exhaustive mapping search completes in approximately one second for a single GEMV, two to three seconds for a single GEMM, and five to ten seconds for an entire LLM workload, on a 16-core consumer-grade CPU. The search space contains 1,548 candidate mappings for a GEMM and 192 for a GEMV; the smaller GEMV space reflects the absence of a meaningful M-dimension and the corresponding reduction in tiling choices. Two factors keep the search tractable: (i) each candidate is evaluated by an analytical model in microseconds, and (ii) the GEMM and GEMV shapes within an LLM are consistent across layers, so per-shape

optimal mappings can be reused. Search overhead can additionally be amortized at deployment: because mapping is data-agnostic, mappings for the discrete set of input token lengths used at runtime can be precomputed or cached, eliminating runtime search cost for repeated shapes.

Integration with High-Level IR. The *DREAM* mapping framework can be integrated with high-level deep-learning frameworks and intermediate representations such as PyTorch [57], TensorFlow [2], MLIR [43], Halide [62], and TVM Relay [9] by annotating PIM-eligible operators and their precision. Within such a pipeline, the framework serves as a mapping pass and a code-generation backend: it lowers operator IR to PIM instructions and inserts the data-layout transformations required by the chosen mapping.

Cost and Total Cost of Ownership. *DREAM*-1 TB consists of commodity DDR5 plus a 1.17% peripheral premium, while a single-GPU system with equivalent capacity is dominated by the GPU die and HBM at substantially higher per-GB cost. The tokens-per-dollar advantage of *DREAM* is therefore on at least the same order as its tokens-per-area advantage in Section 6.2; a precise TCO model is left for future work.

Cross-Rank Broadcast. *DREAM*'s rank-level broadcasters extend the LRDIMM data-buffer (DB) chip with a multi-rank-target write mode that delivers a source rank's column-read output to all peer ranks' locality buffers in one channel transaction. Because the DB already buffers and redrives DQ between host and rank-side, a 1-to-R rank-side fan-out is a small logical generalization of its existing 1-to-1 redrive; nevertheless, three engineering questions remain open: (i) signal-integrity closure of simultaneous multi-rank rank-side writes through the DB has not been demonstrated experimentally; (ii) DB-vendor implementation of a broadcast-write command is required; and (iii) the multi-rank-target write would need a JEDEC command/encoding to be standardized. Our evaluation assumes idealized DB behavior pending these answers; quantifying SI margin and DB-side power on a fabricated prototype is left for future work.

8 RELATED WORK

To the best of our knowledge, *DREAM* is the first DDR-class bit-serial PIM architecture that simultaneously enables bit-level operand reuse and supports a generalized mapping framework that exhaustively explores the mapping space. We position *DREAM* relative to prior PIM work along three dimensions, summarized in Table 2.

Processing-using-DRAM (PUD). A line of prior work [14–16, 18, 20, 21, 39, 45, 46, 55, 59, 65, 66, 82, 88] extends in-DRAM mechanisms such as row-copy and bulk bitwise operations to implement logic via standard DRAM commands [64–66]. PUD systems require no integration of additional processing elements within the array and, in some cases, employ bit-serial schemes to support runtime-variable precision. However, these systems offer limited or no operand reuse and typically rely on hand-tuned or heuristic-based mappings; consequently, prior measurements [14] report poor GEMM performance compared to GPUs on standard benchmarks (*2mm*, *3mm*, and *gmm* from PolyBench [61]).

Digital PIM. PIM architectures with digital PEs are commonly termed digital PIM. SRAM-based systems such as Neural Cache and PIMSAB [17, 52] integrate bit-serial PEs at SRAM arrays to enable in-SRAM computation, but remain bound by DRAM bandwidth at the memory boundary. DRAM-based digital PIM systems [8, 23, 25, 30, 47] integrate near-bank or near-sub-array PEs; their parallelism is bounded by the DRAM column decoder, and most support only a fixed precision, precluding mixed-precision optimization. Other systems [22, 34, 48] extend DRAM with simple logic to accelerate search and lookup primitives but support a narrow class of workloads. *DREAM* adds light-weight bit-serial PEs at array granularity, exploits sub-array-level internal bandwidth, and supports a broad workload class with runtime-variable precision.

Table 2: Capability matrix of prior PIM architectures along the four axes required for end-to-end edge LLM inference: bit-level operand reuse (per-MAC efficiency, see Fig. 1), multi-level broadcasting across the CRBD parallelism hierarchy (mapping flexibility), runtime-variable precision (mixed-precision support), and workload-driven mapping search. *DREAM* is the only design that satisfies all four.

Prior Work	Bit-level operand reuse	Multi-level CRBD broadcast	Runtime-variable precision	Workload-driven mapping search
Neural Cache (SRAM, bit-serial)	✓ (SRAM)	✗	✓	✗
PIMSAB (SRAM, bit-serial)	✓ (SRAM)	partial	✓	heuristic
Newton/AiM (DRAM, bit-parallel)	✗	global bus only	✗ (int8)	✗
SIMDRAM (DRAM, bit-serial)	✗	✗	✓	✗
MIMDRAM (DRAM, bit-serial)	✗	✗	✓	heuristic
Proteus (DRAM, bit-serial)	✗	✗	✓	✗
***DREAM* (Ours)**	**✓ (LB)**	**✓ (B/R/A)**	✓	**✓ (exhaustive)**

Mapping Frameworks. Prior loop-analysis-based mapping frameworks [7, 9, 13, 24, 27, 28, 37, 38, 41, 51, 53, 56, 63, 73, 77–79, 81, 84] target ASIC accelerators and do not generalize directly to in-DRAM substrates, owing to the latter's hierarchical organization and sub-array-aware data-layout constraints. Recent PIM-specific mapping frameworks [11, 49, 87] address these constraints: PIM-DL and ARES employ genetic algorithms; OptiPIM uses integer linear programming to search the mapping space. *DREAM*'s framework instead specializes the search to the GEMM mapping space defined by the DRAM parallelism hierarchy, and combines this restriction with an analytical performance model to enable exhaustive enumeration of the full mapping space within a tractable budget.

9 CONCLUSION

We presented *DREAM*, an in-DRAM bit-serial PIM architecture for single-device LLM inference. *DREAM* resolves the per-MAC and mapping limitations of prior DRAM-PIM through a per-bank locality buffer, a multi-level (bank/rank/array) broadcasting hierarchy, and a generalized analytical mapping framework, delivering 46.3× and 6.4× geometric-mean end-to-end speedups over a single-GPU H100 on decode-dominated and long-context edge inference at 1.17% peripheral-area overhead, while out-performing similarly provisioned PUD and word-PE PIM baselines by geomeans of 112.8× and 5.8×. These results show that per-MAC efficiency and mapping flexibility must be addressed jointly to make DRAM-PIM competitive at the edge.

REFERENCES

[1] 2011. *Synopsys, Inc. Design Compiler Optimization Reference Manual, Version F-2011.09, September 2011.*

[2] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2016. TensorFlow: a system for large-scale machine learning. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation* (Savannah, GA, USA) *(OSDI'16)*. USENIX Association, USA, 265–283.

[3] Reza Yazdani Aminabadi, Samyam Rajbhandari, Minjia Zhang, Ammar Ahmad Awan, Cheng Li, Du Li, Elton Zheng, Jeff Rasley, Shaden Smith, Olatunji Ruwase, and Yuxiong He. 2022. DeepSpeed Inference: Enabling Efficient Inference of Transformer Models at Unprecedented Scale. arXiv:2207.00032 [cs.LG] https://arxiv.org/abs/2207.00032

[4] Aman Arora, Atharva Bhamburkar, Aatman Borda, Tanmay Anand, Rishabh Sehgal, Bagus Hanindhito, Pierre-Emmanuel Gaillardon, Jaydeep Kulkarni, and Lizy K. John. 2023. CoMeFa: Deploying Compute-in-Memory on FPGAs for Deep Learning Acceleration. *ACM Trans. Reconfigurable Technol. Syst.* 16, 3, Article 50 (July 2023), 34 pages. https://doi.org/10.1145/3603504

[5] Himanshu Bhatnagar. 2002. *Advanced ASIC Chip Synthesis: Using Synopsys Design Compiler Physical Compiler and Prime Time* (2nd ed.). Kluwer Academic Publishers, USA.

[6] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Proceedings of the 34th International Conference on Neural Information Processing Systems* (Vancouver, BC, Canada) *(NIPS '20)*. Curran Associates Inc., Red Hook, NY, USA, Article 159, 25 pages.

[7] Prasanth Chatarasi, Hyoukjun Kwon, Angshuman Parashar, Michael Pellauer, Tushar Krishna, and Vivek Sarkar. 2021. Marvel: A Data-Centric Approach for Mapping Deep Learning Operators on Spatial Accelerators. *ACM Trans. Archit. Code Optim.* 19, 1, Article 6 (Dec. 2021), 26 pages. https://doi.org/10.1145/3485137

[8] Liyan Chen, Dongxu Lyu, Zhenyu Li, Jianfei Jiang, Qin Wang, Zhigang Mao, and Naifeng Jing. 2025. AttenPIM: Accelerating LLM Attention with Dual-mode GEMV in Processing-in-Memory. In *2025 62nd ACM/IEEE Design Automation Conference (DAC)*. 1–7. https://doi.org/10.1109/DAC63849.2025.11133230

[9] Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Yan, Meghan Cowan, Haichen Shen, Leyuan Wang, Yuwei Hu, Luis Ceze, Carlos Guestrin, and Arvind Krishnamurthy. 2018. TVM: an automated end-to-end optimizing compiler for deep learning *(OSDI'18)*. USENIX Association, USA, 579–594.

[10] Derek Christ, Lukas Steiner, Matthias Jung, and Norbert Wehn. 2024. PIMSys: A Virtual Prototype for Processing in Memory. In *Proceedings of the International Symposium on Memory Systems (MEMSYS '24)*. Association for Computing Machinery, New York, NY, USA, 26–33. https://doi.org/10.1145/3695794.3695797

[11] Xiuping Cui, Size Zheng, Tianyu Jia, Le Ye, and Yun Liang. 2023. ARES: A Mapping Framework of DNNs Towards Diverse PIMs with General Abstractions. In *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. 1–9. https://doi.org/10.1109/ICCAD57390.2023.10323777

[12] Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness. arXiv:2205.14135 [cs.LG] https://arxiv.org/abs/2205.14135

[13] Shail Dave, Tony Nowatzki, and Aviral Shrivastava. 2024. Explainable-DSE: An Agile and Explainable Exploration of Efficient HW/SW Codesigns of Deep Learning Accelerators Using Bottleneck Analysis. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 4* (Vancouver, BC, Canada) *(ASPLOS '23)*. Association for Computing Machinery, New York, NY, USA, 87–107. https://doi.org/10.1145/3623278.3624772

[14] Geraldo Francisco de Oliveira Junior, Mayank Kabra, Yuxin Guo, Kangqi Chen, Abdullah Giray Yaglikci, Melina Soysal, Mohammad Sadrosadati, Joaquin Olivares Bueno, Saugata Ghose, Juan Gómez-Luna, and Onur Mutlu. 2025. Proteus: Achieving High-Performance Processing-Using-DRAM with Dynamic Bit-Precision, Adaptive Data Representation, and Flexible Arithmetic. In *Proceedings of the 39th ACM International Conference on Supercomputing (ICS '25)*. Association for Computing Machinery, New York, NY, USA, 473–494. https://doi.org/10.1145/3721145.3730420

[15] Quan Deng, Lei Jiang, Youtao Zhang, Minxuan Zhang, and Jun Yang. 2018. DrAcc: a DRAM based Accelerator for Accurate CNN Inference. In *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*. 1–6. https://doi.org/10.1109/DAC.2018.8465866

[16] Quan Deng, Youtao Zhang, Minxuan Zhang, and Jun Yang. 2019. LAcc: Exploiting Lookup Table-based Fast and Accurate Vector Multiplication in DRAM-based CNN Accelerator. In *2019 56th ACM/IEEE Design Automation Conference (DAC)*. 1–6.

[17] Charles Eckert, Xiaowei Wang, Jingcheng Wang, Arun Subramaniyan, Ravi Iyer, Dennis Sylvester, David Blaaauw, and Reetuparna Das. 2018. Neural Cache: Bit-Serial In-Cache Acceleration of Deep Neural Networks. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. 383–396. https://doi.org/10.1109/ISCA.2018.00040

[18] João Dinis Ferreira, Gabriel Falcao, Juan Gómez-Luna, Mohammed Alser, Lois Orosa, Mohammad Sadrosadati, Jeremie S. Kim, Geraldo F. Oliveira, Taha Shahroodi, Anant Nori, and Onur Mutlu. 2022. pLUTo: Enabling Massively Parallel Computation in DRAM via Lookup Tables. In *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 900–919. https://doi.org/10.1109/MICRO56248.2022.00067

[19] Brinda Ganesh, Aamer Jaleel, David Wang, and Bruce Jacob. 2007. Fully-Buffered DIMM Memory Architectures: Understanding Mechanisms, Overheads and Scaling. In *2007 IEEE 13th International Symposium on High Performance Computer Architecture*. 109–120. https://doi.org/10.1109/HPCA.2007.346190

[20] Fei Gao, Georgios Tziantzioulis, and David Wentzlaff. 2019. ComputeDRAM: In-Memory Compute Using Off-the-Shelf DRAMs. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture* (Columbus, OH, USA) *(MICRO-52)*. Association for Computing Machinery, New York, NY, USA, 100–113. https://doi.org/10.1145/3352460.3358260

[21] Nastaran Hajinazar, Geraldo F. Oliveira, Sven Gregorio, João Dinis Ferreira, Nika Mansouri Ghiasi, Minesh Patel, Mohammed Alser, Saugata Ghose, Juan Gómez-Luna, and Onur Mutlu. 2021. 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* (Virtual, USA) *(ASPLOS '21)*. Association for Computing Machinery, New York, NY, USA, 329–345. https://doi.org/10.1145/3445814.3446749

[22] Wontak Han, Hyunjun Cho, Donghyuk Kim, and Joo-Young Kim. 2025. SAL-PIM: A Subarray-Level Processing-in-Memory Architecture With LUT-Based Linear Interpolation for Transformer-Based Text Generation. *IEEE Trans. Comput.* 74, 9 (2025), 2909–2922. https://doi.org/10.1109/TC.2025.3576935

[23] Mingxuan He, Choungki Song, Ilkon Kim, Chunseok Jeong, Seho Kim, Il Park, Mithuna Thottethodi, and T. N. Vijaykumar. 2020. Newton: A DRAM-maker's Accelerator-in-Memory (AiM) Architecture for Machine Learning. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 372–385. https://doi.org/10.1109/MICRO50266.2020.00040

[24] Kartik Hegde, Po-An Tsai, Sitao Huang, Vikas Chandra, Angshuman Parashar, and Christopher W. Fletcher. 2021. Mind mappings: enabling efficient algorithm-accelerator mapping space search. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems* (Virtual, USA) *(ASPLOS '21)*. Association for Computing Machinery, New York, NY, USA, 943–958. https://doi.org/10.1145/3445814.3446762

[25] Guseul Heo, Sangyeop Lee, Jaehong Cho, Hyunmin Choi, Sanghyeon Lee, Hyungkyu Ham, Gwangsun Kim, Divya Mahajan, and Jongse Park. 2024. 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, Volume 3* (La Jolla, CA, USA) *(ASPLOS '24)*. Association for Computing Machinery, New York, NY, USA, 722–737. https://doi.org/10.1145/3620666.3651380

[26] Seongmin Hong, Seungjae Moon, Junsoo Kim, Sungjae Lee, Minsub Kim, Dongsoo Lee, and Joo-Young Kim. 2022. DFX: A Low-latency Multi-FPGA Appliance for Accelerating Transformer-based Text Generation. In *2022 IEEE Hot Chips 34 Symposium (HCS)*. 1–17. https://doi.org/10.1109/HCS55958.2022.9895626

[27] Mark Horeni, Pooria Taheri, Po-An Tsai, Angshuman Parashar, Joel Emer, and Siddharth Joshi. 2022. Ruby: Improving Hardware Efficiency for Tensor Algebra Accelerators Through Imperfect Factorization. In *2022 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. 254–266. https://doi.org/10.1109/ISPASS55109.2022.00039

[28] Qijing Huang, Minwoo Kang, Grace Dinh, Thomas Norell, Aravind Kalaiah, James Demmel, John Wawrzynek, and Yakun Sophia Shao. 2021. CoSA: scheduling by <u>c</u>onstrained <u>o</u>ptimization for <u>s</u>patial <u>a</u>ccelerators. In *Proceedings of the 48th Annual International Symposium on Computer Architecture* (Virtual Event, Spain) *(ISCA '21)*. IEEE Press, 554–566. https://doi.org/10.1109/ISCA52012.2021.00050

[29] Wei Huang, Haotong Qin, Yangdong Liu, Yawei Li, Qinshuo Liu, Xianglong Liu, Luca Benini, Michele Magno, Shiming Zhang, and Xiaojuan Qi. 2025. SliM-LLM: Salience-Driven Mixed-Precision Quantization for Large Language Models. arXiv:2405.14917 [cs.LG] https://arxiv.org/abs/2405.14917

[30] Jungi Hyun, Minseok Seo, Seongho Jeong, Hyuk-Jae Lee, and Xuan Truong Nguyen. 2025. DEAR-PIM: Processing-in-Memory Architecture with Disaggregated Execution of All-bank Requests. In *2025 Design, Automation & Test in Europe Conference (DATE)*. 1–7. https://doi.org/10.23919/DATE64628.2025.10992691

[31] Imec. 2020. Technology platform for thermally stable DRAM peripheral transistors. *Imec Magazine* (2020). https://www.imec-int.com/en/articles/technology-platform-thermally-stable-dram-peripheral-transistors Accessed: 2025-09-28.

[32] Mikhail Isaev, Nic Mcdonald, Larry Dennison, and Richard Vuduc. 2023. Calculon: a methodology and tool for high-level co-design of systems and large

language models. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (Denver, CO, USA) *(SC '23)*. Association for Computing Machinery, New York, NY, USA, Article 71, 14 pages. https://doi.org/10.1145/3581784.3607102

[33] Bruce Jacob, Spencer Ng, and David Wang. 2007. *Memory Systems: Cache, DRAM, Disk*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[34] Yoonho Jang, Hyeongjun Cho, Yesin Ryu, Jungrae Kim, and Seokin Hong. 2025. PIMPAL: Accelerating LLM Inference on Edge Devices via In-DRAM Arithmetic Lookup. In *2025 62nd ACM/IEEE Design Automation Conference (DAC)*. 1–7. https://doi.org/10.1109/DAC63849.2025.11133391

[35] Alex Kahng, Andrew B. Kahng, Hyein Lee, and Jiajia Li. 2018. PROBE: A Placement, Routing, Back-End-of-Line Measurement Utility. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 37, 7 (2018), 1459–1472. https://doi.org/10.1109/TCAD.2017.2750072

[36] Andrew B. Kahng and et al. 2022. PROBE2.0: A Systematic Framework for Routability/BEOL Co-Optimization. *IEEE TCAD* (2022).

[37] Sheng-Chun Kao and Tushar Krishna. 2020. GAMMA: Automating the HW Mapping of DNN Models on Accelerators via Genetic Algorithm. In *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. 1–9.

[38] Sheng-Chun Kao, Michael Pellauer, Angshuman Parashar, and Tushar Krishna. 2022. DiGamma: domain-aware genetic algorithm for HW-mapping co-optimization for DNN accelerators. In *Proceedings of the 2022 Conference & Exhibition on Design, Automation & Test in Europe* (Antwerp, Belgium) *(DATE '22)*. European Design and Automation Association, Leuven, BEL, 232–237.

[39] Jeremie S. Kim, Minesh Patel, Hasan Hassan, Lois Orosa, and Onur Mutlu. 2019. D-RaNGe: Using Commodity DRAM Devices to Generate True Random Numbers with Low Latency and High Throughput. In *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 582–595. https://doi.org/10.1109/HPCA.2019.00011

[40] Yoongu Kim, Weikun Yang, and Onur Mutlu. 2016. Ramulator: A Fast and Extensible DRAM Simulator. *IEEE Computer Architecture Letters* 15, 1 (2016), 45–49. https://doi.org/10.1109/LCA.2015.2414456

[41] Hyoukjun Kwon, Ananda Samajdar, and Tushar Krishna. 2018. MAERI: Enabling Flexible Dataflow Mapping over DNN Accelerators via Reconfigurable Interconnects. *SIGPLAN Not.* 53, 2 (March 2018), 461–475. https://doi.org/10.1145/3296957.3173176

[42] Yongkee Kwon, Kornijcuk Vladimir, Nahsung Kim, Woojae Shin, Jongsoon Won, Minkyu Lee, Hyunha Joo, Haerang Choi, Guhyun Kim, Byeongju An, Jeongbin Kim, Jaewook Lee, Ilkon Kim, Jaehan Park, Chanwook Park, Yosub Song, Byeongsu Yang, Hyungdeok Lee, Seho Kim, Daehan Kwon, Seongju Lee, Kyuyoung Kim, Sanghoon Oh, Joonhong Park, Gimoon Hong, Dongyoon Ka, Kyudong Hwang, Jeongje Park, Kyeongpil Kang, Jungyeon Kim, Junyeol Jeon, Myeongjun Lee, Minyoung Shin, Minhwan Shin, Jaekyung Cha, Changson Jung, Kijoon Chang, Chunseok Jeong, Euicheol Lim, Il Park, Junhyun Chun, and Sk Hynix. 2022. System Architecture and Software Stack for GDDR6-AiM. In *2022 IEEE Hot Chips 34 Symposium (HCS)*. 1–25. https://doi.org/10.1109/HCS55958.2022.9895629

[43] Chris Lattner, Mehdi Amini, Uday Bondhugula, Albert Cohen, Andy Davis, Jacques Pienaar, River Riddle, Tatiana Shpeisman, Nicolas Vasilache, and Oleksandr Zinenko. 2021. MLIR: scaling compiler infrastructure for domain specific computation. In *Proceedings of the 2021 IEEE/ACM International Symposium on Code Generation and Optimization* (Virtual Event, Republic of Korea) *(CGO '21)*. IEEE Press, 2–14. https://doi.org/10.1109/CGO51591.2021.9370308

[44] Cong Li, Yihan Yin, Xintong Wu, Jingchen Zhu, Zhutianya Gao, Dimin Niu, Qiang Wu, Xin Si, Yuan Xie, Chen Zhang, and Guangyu Sun. 2025. H2-LLM: Hardware-Dataflow Co-Exploration for Heterogeneous Hybrid-Bonding-based Low-Batch LLM Inference. In *Proceedings of the 52nd Annual International Symposium on Computer Architecture (ISCA '25)*. Association for Computing Machinery, New York, NY, USA, 194–210. https://doi.org/10.1145/3695053.3731008

[45] Shuangchen Li, Alvin Oliver Glova, Xing Hu, Peng Gu, Dimin Niu, Krishna T. Malladi, Hongzhong Zheng, Bob Brennan, and Yuan Xie. 2018. SCOPE: A Stochastic Computing Engine for DRAM-Based In-Situ Accelerator. In *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 696–709. https://doi.org/10.1109/MICRO.2018.00062

[46] Shuangchen Li, Dimin Niu, Krishna T. Malladi, Hongzhong Zheng, Bob Brennan, and Yuan Xie. 2017. DRISA: A DRAM-based Reconfigurable In-Situ Accelerator. In *2017 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 288–301.

[47] Zhichun Li, Jun Zhou, Xueqi Li, and Ninghui Sun. 2025. BlockPIM: Optimizing Memory Management for PIM-enabled Long-Context LLM Inference. In *2025 62nd ACM/IEEE Design Automation Conference (DAC)*. 1–7. https://doi.org/10.1109/DAC63849.2025.11133193

[48] Chaoqiang Liu, Dan Chen, Yu Huang, Wenjing Xiao, Haifeng Liu, Yi Zhang, Huize Li, Xiaofei Liao, and Hai Jin. 2025. SeIM: In-Memory Acceleration for Approximate Nearest Neighbor Search. In *2025 62nd ACM/IEEE Design Automation Conference (DAC)*. 1–7. https://doi.org/10.1109/DAC63849.2025.11132620

[49] Jiantao Liu, Minxuan Zhou, Yue Pan, Chien-Yi Yang, Lana Josipović, and Tajana Rosing. 2025. OptiPIM: Optimizing Processing-in-Memory Acceleration Using Integer Linear Programming. In *Proceedings of the 52nd Annual International Symposium on Computer Architecture (ISCA '25)*. Association for Computing Machinery, New York, NY, USA, 867–883. https://doi.org/10.1145/3695053.3731041

[50] Mukul Lokhande, Gopal Raut, and Santosh Kumar Vishvakarma. 2025. Flex-PE: Flexible and SIMD Multiprecision Processing Element for AI Workloads. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 33, 6 (2025), 1610–1623. https://doi.org/10.1109/TVLSI.2025.3553069

[51] Liqiang Lu, Naiqing Guan, Yuyue Wang, Liancheng Jia, Zizhang Luo, Jieming Yin, Jason Cong, and Yun Liang. 2021. TENET: A Framework for Modeling Tensor Dataflow Based on Relation-centric Notation. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. 720–733. https://doi.org/10.1109/ISCA52012.2021.00062

[52] Siyuan Ma, Kaustubh Mhatre, Jian Weng, Bagus Hanindhito, Zhengrong Wang, Tony Nowatzki, Lizy John, and Aman Arora. 2024. PIMSAB: A Processing-In-Memory System with Spatially-Aware Communication and Bit-Serial-Aware Computation. *ACM Trans. Archit. Code Optim.* 21, 4, Article 91 (Nov. 2024), 27 pages. https://doi.org/10.1145/3690824

[53] Wei Niu, Gagan Agrawal, and Bin Ren. 2024. SoD2: Statically Optimizing Dynamic Deep Neural Network Execution. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1* (La Jolla, CA, USA) *(ASPLOS '24)*. Association for Computing Machinery, New York, NY, USA, 386–400. https://doi.org/10.1145/3617232.3624869

[54] NVIDIA. 2025. NVIDIA H100 Tensor Core GPU Architecture. https://resources.nvidia.com/en-us-hopper-architecture/nvidia-h100-tensor-c. Accessed: 17 November 2025.

[55] Geraldo F. Oliveira, Ataberk Olgun, Abdullah Giray Yağlıkçı, F. Nisa Bostancı, Juan Gómez-Luna, Saugata Ghose, and Onur Mutlu. 2024. MIMDRAM: An End-to-End Processing-Using-DRAM System for High-Throughput, Energy-Efficient and Programmer-Transparent Multiple-Instruction Multiple-Data Computing. In *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. 186–203. https://doi.org/10.1109/HPCA57654.2024.00024

[56] Angshuman Parashar, Priyanka Raina, Yakun Sophia Shao, Yu-Hsin Chen, Victor A. Ying, Anurag Mukkara, Rangharajan Venkatesan, Brucek Khailany, Stephen W. Keckler, and Joel Emer. 2019. Timeloop: A Systematic Approach to DNN Accelerator Evaluation. In *2019 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. 304–315. https://doi.org/10.1109/ISPASS.2019.00042

[57] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. *PyTorch: an imperative style, high-performance deep learning library*. Curran Associates Inc., Red Hook, NY, USA.

[58] Arvind Pathak, Rajeev Balasubramonian, and Norm Jouppi. 2013. A modeling methodology for DRAM with 3D integration and heterogeneous technologies. In *Proceedings of the Design, Automation and Test in Europe Conference (DATE)*. IEEE, 1549–1554. https://repository.lib.ncsu.edu/bitstreams/004e6091-4d63-41e7-9bc6-1d8a15cf3b22/download

[59] Xiangjun Peng, Yaohua Wang, and Ming-Chang Yang. 2023. CHOPPER: A Compiler Infrastructure for Programmable Bit-serial SIMD Processing Using Memory in DRAM. In *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. 1275–1288. https://doi.org/10.1109/HPCA56546.2023.10071070

[60] Reiner Pope, Sholto Douglas, Aakanksha Chowdhery, Jacob Devlin, James Bradbury, Anselm Levskaya, Jonathan Heek, Kefan Xiao, Shivani Agrawal, and Jeff Dean. 2022. Efficiently Scaling Transformer Inference. arXiv:2211.05102 [cs.LG] https://arxiv.org/abs/2211.05102

[61] L.-N. Pouchet. 2025. PolyBench: The Polyhedral Benchmark Suite. https://www.cs.colostate.edu/~pouchet/software/polybench/ Accessed on Nov 16, 2025.

[62] Jonathan Ragan-Kelley, Connelly Barnes, Andrew Adams, Sylvain Paris, Frédo Durand, and Saman Amarasinghe. 2013. Halide: a language and compiler for optimizing parallelism, locality, and recomputation in image processing pipelines. In *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation* (Seattle, Washington, USA) *(PLDI '13)*. Association for Computing Machinery, New York, NY, USA, 519–530. https://doi.org/10.1145/2491956.2462176

[63] Enrico Russo, Maurizio Palesi, Giuseppe Ascia, Davide Patti, Salvatore Monteleone, and Vincenzo Catania. 2023. Memory-Aware DNN Algorithm-Hardware Mapping via Integer Linear Programming. In *Proceedings of the 20th ACM International Conference on Computing Frontiers* (Bologna, Italy) *(CF '23)*. Association for Computing Machinery, New York, NY, USA, 134–143. https://doi.org/10.1145/3587135.3592206

[64] Vivek Seshadri, Kevin Hsieh, Amirali Boroum, Donghyuk Lee, Michael A. Kozuch, Onur Mutlu, Phillip B. Gibbons, and Todd C. Mowry. 2015. Fast Bulk Bitwise AND and OR in DRAM. *IEEE Computer Architecture Letters* 14, 2 (2015), 127–131. https://doi.org/10.1109/LCA.2015.2434872

[65] Vivek Seshadri, Yoongu Kim, Chris Fallin, Donghyuk Lee, Rachata Ausavarungnirun, Gennady Pekhimenko, Yixin Luo, Onur Mutlu, Phillip B. Gibbons, Michael A. Kozuch, and Todd C. Mowry. 2013. RowClone: Fast and energy-efficient in-DRAM bulk data copy and initialization. In *2013 46th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 185–197.

[66] Vivek Seshadri, Donghyuk Lee, Thomas Mullins, Hasan Hassan, Amirali Boroumand, Jeremie Kim, Michael A. Kozuch, Onur Mutlu, Phillip B. Gibbons, and Todd C. Mowry. 2017. Ambit: In-Memory Accelerator for Bulk Bitwise Operations Using Commodity DRAM Technology. In *2017 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 273–287.

[67] Wongyu Shin, Jaemin Jang, Jungwhan Choi, Jinwoong Suh, Yongkee Kwon, Youngsuk Moon, and Lee-Sup Kim. 2017. Rank-Level Parallelism in DRAM. *IEEE Trans. Comput.* 66, 7 (2017), 1274–1280. https://doi.org/10.1109/TC.2017.2654339

[68] Cristina Silvano, Daniele Ielmini, Fabrizio Ferrandi, Leandro Fiorin, Serena Curzel, Luca Benini, Francesco Conti, Angelo Garofalo, Cristian Zambelli, Enrico Calore, Sebastiano Schifano, Maurizio Palesi, Giuseppe Ascia, Davide Patti, Nicola Petra, Davide De Caro, Luciano Lavagno, Teodoro Urso, Valeria Cardellini, Gian Carlo Cardarilli, Robert Birke, and Stefania Perri. 2025. A Survey on Deep Learning Hardware Accelerators for Heterogeneous HPC Platforms. 57, 11, Article 286 (June 2025), 39 pages. https://doi.org/10.1145/3729215

[69] Alessandro Spessot, Thomas Machleidt, Colleen Kenyon, and Chia-Hung Tsai. 2015. Optimized material solutions for advanced DRAM. *Physica Status Solidi (a)* 212, 1 (2015), 47–53. https://doi.org/10.1002/pssa.201532791

[70] Peter Spindler and Frank M. Johannes. 2007. Fast and Accurate Routing Demand Estimation for Efficient Routability-Driven Placement. In *DATE*.

[71] EDN Staff. 2018. A new platform for thermally stable DRAM peripheral transistors. *EDN Network* (2018). https://www.edn.com/a-new-platform-for-thermally-stable-dram-peripheral-transistors Accessed: 2025-09-28.

[72] James E. Stine, Ivan Castellanos, Michael Wood, Jeff Henson, Fred Love, W. Rhett Davis, Paul D. Franzon, Michael Bucher, Sunil Basavarajaiah, Julie Oh, and Ravi Jenkal. 2007. FreePDK: An Open-Source Variation-Aware Design Kit. In *2007 IEEE International Conference on Microelectronic Systems Education (MSE'07)*. 173–174. https://doi.org/10.1109/MSE.2007.44

[73] Zhanhong Tan, Zijian Zhu, and Kaisheng Ma. 2024. Cocco: Hardware-Mapping Co-Exploration towards Memory Capacity-Communication Optimization. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1* (La Jolla, CA, USA) *(ASPLOS '24)*. Association for Computing Machinery, New York, NY, USA, 69–84. https://doi.org/10.1145/3617232.3624865

[74] Llama Team. 2024. The Llama 3 Herd of Models. arXiv:2407.21783 [cs.AI] https://arxiv.org/abs/2407.21783

[75] TechInsights. 2022. *Industry-leading DDR5 Technology: Micron vs. Samsung*. https://www.techinsights.com/blog/industry-leading-ddr5-technology Accessed: 2025-10-02.

[76] Donald Thomas and Philip Moorby. 2008. *The Verilog Hardware Description Language* (5th ed. ed.). Springer Publishing Company, Incorporated.

[77] Philippe Tillet, H. T. Kung, and David Cox. 2019. Triton: an intermediate language and compiler for tiled neural network computations. In *Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages* (Phoenix, AZ, USA) *(MAPL 2019)*. Association for Computing Machinery, New York, NY, USA, 10–19. https://doi.org/10.1145/3315508.3329973

[78] Nicolas Vasilache, Oleksandr Zinenko, Theodoros Theodoridis, Priya Goyal, Zachary DeVito, William S. Moses, Sven Verdoolaege, Andrew Adams, and Albert Cohen. 2018. Tensor Comprehensions: Framework-Agnostic High-Performance Machine Learning Abstractions. arXiv:1802.04730 [cs.PL] https://arxiv.org/abs/1802.04730

[79] Jie Wang and Jason Cong. 2021. Search for Optimal Systolic Arrays: A Comprehensive Automated Exploration Framework and Lessons Learned. arXiv:2111.14252 [cs.AR] https://arxiv.org/abs/2111.14252

[80] Ke Wang, Kevin Angstadt, Chunkun Bo, Nathan Brunelle, Elaheh Sadredini, II Tracy, Tommy, Jack Wadden, Mircea Stan, and Kevin Skadron. 2016. An overview of micron's automata processor. In *Proceedings of the Eleventh IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis* (Pittsburgh, Pennsylvania) *(CODES '16)*. Association for Computing Machinery, New York, NY, USA, Article 14, 3 pages. https://doi.org/10.1145/2968456.2976763

[81] Yannan Nellie Wu, Po-An Tsai, Angshuman Parashar, Vivienne Sze, and Joel S. Emer. 2023. Sparseloop: An Analytical Approach to Sparse Tensor Accelerator Modeling. In *Proceedings of the 55th Annual IEEE/ACM International Symposium on Microarchitecture* (Chicago, Illinois, USA) *(MICRO '22)*. IEEE Press, 1377–1395. https://doi.org/10.1109/MICRO56248.2022.00096

[82] Xin Xin, Youtao Zhang, and Jun Yang. 2020. ELP2IM: Efficient and Low Power Bitwise Operation Processing in DRAM. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 303–314. https://doi.org/10.1109/HPCA47549.2020.00033

[83] Fu-Liang Yang, Cheng-Chuan Huang, Chien-Chao Huang, Tang-Xuan Chung, Hou-Yu Chen, Chang-Yun Chang, Hung-Wei Chen, Di-Hong Lee, Sheng-Da Liu, Kuang-Hsin Chen, Cheng-Kuo Wen, Shui-Ming Cheng, Chang-Ta Yang, Li-Wei Kung, Chiu-Lien Lee, Yu-Jun Chou, Fu-Jye Liang, Lin-Hung Shiu, Jan-Wen You, King-Chang Shu, Bin-Chang Chang, Jaw-Jung Shin, Chun-Kuang Chen, Tsai-Sheng Gau, Ping-Wei Wang, Bor-Wen Chan, Peng-Fu Hsu, Jyu-Honig Shieh, S.K.-H. Fung, C.H. Diaz, C.-M.M. Wu, Yee-Chaung See, B.J. Lin, M.-S. Liang, J.Y.-C. Sun, and Chenming Hu. 2004. 45nm node planar-SOI technology with 0.296 /spl mu/m/sup 2/ 6T-SRAM cell. In *Digest of Technical Papers. 2004 Symposium on VLSI Technology, 2004.* 8–9. https://doi.org/10.1109/VLSIT.2004.1345362

[84] Xuan Yang, Mingyu Gao, Qiaoyi Liu, Jeff Setter, Jing Pu, Ankita Nayak, Steven Bell, Kaidi Cao, Heonjae Ha, Priyanka Raina, Christos Kozyrakis, and Mark Horowitz. 2020. Interstellar: Using Halide's Scheduling Language to Analyze DNN Accelerators. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems* (Lausanne, Switzerland) *(ASPLOS '20)*. Association for Computing Machinery, New York, NY, USA, 369–383. https://doi.org/10.1145/3373376.3378514

[85] Haichao Yu, Haoxiang Li, Honghui Shi, Thomas S. Huang, and Gang Hua. 2021. Any-Precision Deep Neural Networks. arXiv:1911.07346 [cs.LG] https://arxiv.org/abs/1911.07346

[86] Hengrui Zhang, August Ning, Rohan Baskar Prabhakar, and David Wentzlaff. 2025. LLMCompass: Enabling Efficient Hardware Design for Large Language Model Inference. In *Proceedings of the 51st Annual International Symposium on Computer Architecture* (Buenos Aires, Argentina) *(ISCA '24)*. IEEE Press, 1080–1096. https://doi.org/10.1109/ISCA59077.2024.00082

[87] Minxuan Zhou, Guoyang Chen, Mohsen Imani, Saransh Gupta, Weifeng Zhang, and Tajana Rosing. 2021. PIM-DL: Boosting DNN Inference on Digital Processing In-Memory Architectures via Data Layout Optimizations. In *2021 30th International Conference on Parallel Architectures and Compilation Techniques (PACT)*. 1–1. https://doi.org/10.1109/PACT52795.2021.00021

[88] Ranyang Zhou, Arman Roohi, Durga Misra, and Shaahin Angizi. 2022. FlexiDRAM: A Flexible in-DRAM Framework to Enable Parallel General-Purpose Computation *(ISLPED '22)*. Association for Computing Machinery, New York, NY, USA, Article 7, 6 pages. https://doi.org/10.1145/3531437.3539721