Breaking the Memory Wall: A Survey of DRAM-based Processing-In-Memory Architectures and Systems

QIUSHI LIN*, Tsinghua University, Beijing, China
ZHENHUA ZHU*†, Tsinghua University, Beijing, China
TONGXIN XIE, Tsinghua University, Beijing, China
YIWEI ZHAO, Carnegie Mellon University, Pittsburgh, PA, USA
GUY E. BLELLOCH, Carnegie Mellon University, Pittsburgh, PA, USA
PHILLIP B. GIBBONS, Carnegie Mellon University, Pittsburgh, PA, USA
GUOHAO DAI, Shanghai Jiao Tong University, Shanghai, China
MINGYU GAO, Tsinghua University, Beijing, China
YUAN XIE, Hong Kong University of Science and Technology, Hong Kong, China
YU WANG, Tsinghua University, Beijing, China

In the big-data and AI era, algorithmic performance improve predominantly by scaling the data volume and computation amounts. Yet under the traditional von Neumann separation of compute and storage, moving data across the memory hierarchy and processing unit has been more and more expensive, causing severe "memory wall" problem. Processing-in-memory (PIM) architectures, which put processing units near/within memory arrays, show the promising potential to solve this problem by collapsing data movement and amplifying effective bandwidth. Among different PIM implementations, DRAM-based PIM stands out as a practical path to scale considering its high storage density with a mature manufacturing. In this article, we provide a comprehensive survey of DRAM-based PIM, categorizing architectures across different integration levels—from bank-level to DIMM-level—and execution paradigms. Beyond hardware, we systematically analyze the requisite system extensions, including programming models, OS management, and coherence mechanisms, alongside the simulation tools essential for performance evaluation. Finally, we identify critical open challenges, such as security and thermal reliability, and outline future directions toward standardization and CXL-enabled integration.

CCS Concepts: • **Do Not Use This Code → Generate the Correct Terms for Your Paper**; *Generate the Correct Terms for Your Paper*; Generate the Correct Terms for Your Paper; Generate the Correct Terms for Your Paper.

Additional Key Words and Phrases: DRAM, Processing-In-Memory, Near-Memory-Computing, Near-Data-Processing

---

*Both authors contributed equally to this research.
†Corresponding author.

Authors' Contact Information: Qiushi Lin, Tsinghua University, Beijing, China; Zhenhua Zhu, Tsinghua University, Beijing, China; Tongxin Xie, Tsinghua University, Beijing, China; Yiwei Zhao, Carnegie Mellon University, Pittsburgh, PA, USA; Guy E. Blelloch, Carnegie Mellon University, Pittsburgh, PA, USA; Phillip B. Gibbons, Carnegie Mellon University, Pittsburgh, PA, USA; Guohao Dai, Shanghai Jiao Tong University, Shanghai, China; Mingyu Gao, Tsinghua University, Beijing, China; Yuan Xie, Hong Kong University of Science and Technology, Hong Kong, China; Yu Wang, Tsinghua University, Beijing, China.

ACM 1557-735X/2018/8-ART111
https://doi.org/XXXXXXX.XXXXXXX

ACM Reference Format:
Qiushi Lin, Zhenhua Zhu, Tongxin Xie, Yiwei Zhao, Guy E. Blelloch, Phillip B. Gibbons, Guohao Dai, Mingyu Gao, Yuan Xie, and Yu Wang. 2018. Breaking the Memory Wall: A Survey of DRAM-based Processing-In-Memory Architectures and Systems. *J. ACM* 37, 4, Article 111 (August 2018), 52 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

In the era of data intensive computing and large scale AI, system performance is increasingly constrained not by arithmetic throughput, but by the cost of moving data between processors and memory. As processor microarchitectures and specialized accelerators (GPUs, TPUs, NPUs) continue to scale in parallelism, the bandwidth and latency of off chip memory interfaces have failed to keep pace. This long standing memory wall manifests most severely in workloads that are both data hungry and memory bound, such as deep neural networks and large language models, graph analytics and graph neural networks, large scale recommendation systems, database and analytics engines, and fully homomorphic encryption. In these applications, energy and latency are dominated by memory traffic rather than computation, which makes further performance gains increasingly difficult under the traditional von Neumann separation of compute and storage.

Processing in Memory (PIM) has emerged as a promising paradigm to address this bottleneck by placing compute capability near or within memory devices, which reduces data movement and exposes the enormous internal bandwidth of modern DRAM systems. Within the broad spectrum of PIM approaches, DRAM based PIM stands out as a particularly practical and scalable direction. DRAM offers high density, mature manufacturing, and established standards (DDR, LPDDR, GDDR, HBM, HMC), and many recent research prototypes and industrial products already build on commodity or lightly modified DRAM devices. At the same time, DRAM technology itself is complex, with hierarchical organizations (channel, rank, chip, bank group, bank, subarray), stringent timing constraints, refresh mechanisms, and diverse device families that all interact with PIM design choices.

The design space of DRAM-PIM has rapidly expanded along several axes: where computation is placed relative to the DRAM cell array (from in array operations to near bank, rank or DIMM level, and 3D stacked designs); how PIM units are integrated into larger systems (communication fabrics, host–PIM coordination, scale up and scale out); how they are exposed to programmers (simulation frameworks, programming models, compilers, and runtimes); and how software algorithms and data structures are rethought to exploit DRAM centric execution while maintaining security and reliability. This diversity calls for a structured view that connects device physics, architecture, system software, and application co design within a unified DRAM centered perspective.

This survey aims to provide such a view by focusing specifically on DRAM based PIM architectures and systems. We begin with the necessary background on DRAM fundamentals and representative memory intensive workloads and use the memory wall problem to motivate why DRAM-PIM is particularly compelling for today's AI and data analytics landscape (Section 2). We then introduce a general taxonomy of PIM paradigms and use it to narrow the scope to DRAM based PIM, distinguishing it from analog or non volatile compute in memory approaches while emphasizing compatibility with commodity DRAM manufacturing (Section 2.3).

Building on this foundation, the core of the survey follows a bottom up logic. Section 3 first categorizes DRAM-PIM hardware according to the physical proximity of compute to the DRAM array: Processing using DRAM (true in array operations that repurpose sense amplifiers and bitlines), Processing near Bank (lightweight compute engines in bank peripheries), Rank or DIMM level PIM (more powerful logic in buffer chips without changing DRAM devices), and 3D hybrid bonding based PIM (logic in stack architectures with ultra dense vertical links). For each category, we discuss core ideas, strengths and weaknesses, and representative designs, and conclude with a comparative view that highlights key trade offs in bandwidth, parallelism, programmability, and hardware invasiveness.

Hardware alone is insufficient to realize the promise of DRAM-PIM. Section 4 therefore moves up one level to system integration and examines how individual PIM units are composed into usable systems. We discuss inter PIM communication (software and hardware level fabrics), host–PIM coordination (overlapping execution and resolving memory space contention), integration with heterogeneous devices such as CPUs and GPUs inside a node, and scale out over fabrics such as CXL for disaggregated memory. This section emphasizes that PIM benefits depend critically on system level design choices rather than on device capabilities in isolation.

Section 5 then focuses on the software stack that is required to design and utilize DRAM-PIM systems. We review simulation and evaluation frameworks that capture both DRAM timing and in memory compute, programming models and compilers that decide what to offload and where, runtime systems that dynamically schedule and migrate tasks and data, and design space exploration tools that jointly reason about compute organization, DRAM structure, and workload mapping. Together, these tools form the bridge between abstract applications and concrete DRAM-PIM hardware and enable both architects and application developers to reason about performance, energy, and scalability.

Section 6 takes a complementary, application driven perspective on software and DRAM-PIM co design. We summarize domain specific practices across deep learning (from CNNs to LLMs), recommendation systems, privacy preserving computation (FHE), graph analytics, databases, and other data intensive workloads. In each domain, we highlight how algorithms and data structures are restructured to align with DRAM organization, bank level parallelism, and near data compute primitives, and how host–PIM collaboration is orchestrated for both performance and capacity. Section 7 then discusses cross cutting issues in security and reliability, including disturbance effects similar to RowHammer under PIM access patterns, covert and side channels, device trust, and secure offloading mechanisms, which are becoming first class concerns as PIM moves from research prototypes to industrial deployment. Section **??** concludes with open challenges and future directions, and Section 9 provides a consolidated application summary.

In summary, this survey presents a coherent logical path that starts from DRAM technology and workload demands, continues through architectural taxonomy and system integration, and extends to software stacks, algorithm co design, and security. The goal is to guide both researchers and practitioners in understanding, evaluating, and advancing DRAM based Processing in Memory systems.

2 Background

This section establishes the foundational knowledge necessary for understanding DRAM-based PIM architectures. We specifically focus on three key dimensions: the fundamental organization and operation of DRAM systems, the characteristics of memory-intensive workloads that necessitate PIM acceleration, and a taxonomy of general PIM paradigms. We begin by detailing the hierarchical structure and timing constraints of modern DRAM to highlight the physical limitations of current memory devices. Subsequently, we analyze the memory wall bottleneck in representative applications to clarify the motivation behind PIM. Finally, we classify different PIM approaches to distinguish DRAM-based solutions from other in-memory computing techniques. This background equips readers with the essential context to comprehend the design challenges and architectural trade-offs discussed in the following sections.

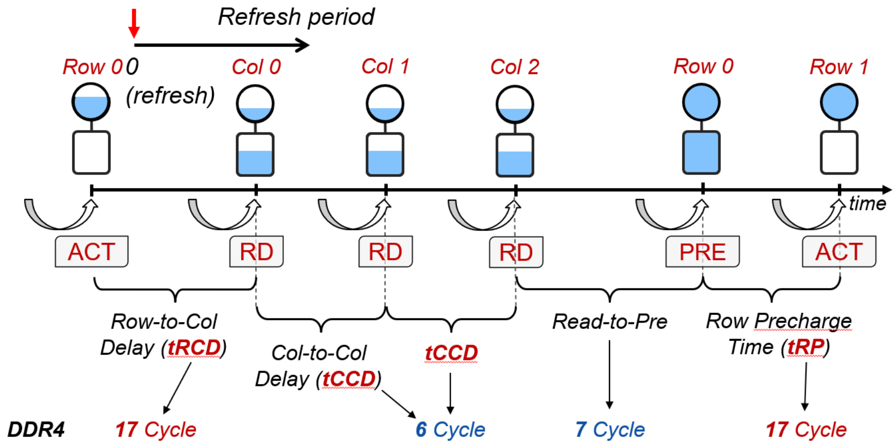

Fig. 1. Timing diagram of standard DRAM operations (Activate, Read/Write, Precharge).

2.1 DRAM Fundamentals

2.1.1 DRAM Working Principle[116]. The fundamental building block of DRAM is the dynamic storage element, commonly implemented as a **1T1C structure** consisting of one access transistor and one capacitor. The capacitor stores a binary state (logic '1' or '0') as electric charge, while the access transistor connects the capacitor to the bitline under the control of the wordline. To access data, as Figure 1 shows, the memory controller follows a strict command sequence: *Activate*, *Read/Write*, and *Precharge*. First, an ***Activate*** command opens a target row by asserting the wordline, connecting all cells in that row to their corresponding bitlines. Subsequently, *Read* or *Write* commands are issued to access specific columns within the open row. Finally, a ***Precharge*** command must be issued to close the active row and restore bitlines to their reference voltage (typically $V_{DD}/2$), preparing the bank for future accesses.

A critical characteristic of the 1T1C cell is **charge leakage**. Over time, the stored charge dissipates; if it drops below a certain threshold, the binary value becomes ambiguous, making it difficult to distinguish between logic '0' and '1'. To prevent data loss, the memory controller must periodically issue ***refresh*** commands, which effectively **re-read and rewrite** the data in each row to restore the capacitor charge. Although essential for data integrity, this mechanism introduces significant overheads. Frequent refresh cycles increase power consumption and reduce effective memory bandwidth by temporarily blocking normal read and write accesses. Furthermore, modern DRAM devices face escalating reliability challenges. Beyond charge leakage, issues such as the ***RowHammer*** effect and other **soft or hard errors** necessitate even more robust refresh management. Despite these drawbacks, the refresh mechanism remains indispensable for maintaining the reliability of DRAM—a **cost-efficient yet inherently dynamic** technology. Understanding these constraints is vital, as many PIM architectures seek to either mitigate these overheads or opportunistically utilize the internal refresh bandwidth.

2.1.2 Hierarchical Structure of Modern DRAM Systems. At the top level, the memory controller communicates with DRAM modules via a **Channel**, which provides the necessary command, address, and data buses. While systems often utilize *Dual Inline Memory Modules* (DIMMs) for physical packaging, the logical hierarchy begins with the **Rank**. A channel controls one or more ranks, where each rank consists of multiple DRAM chips operating in lockstep to fill the data

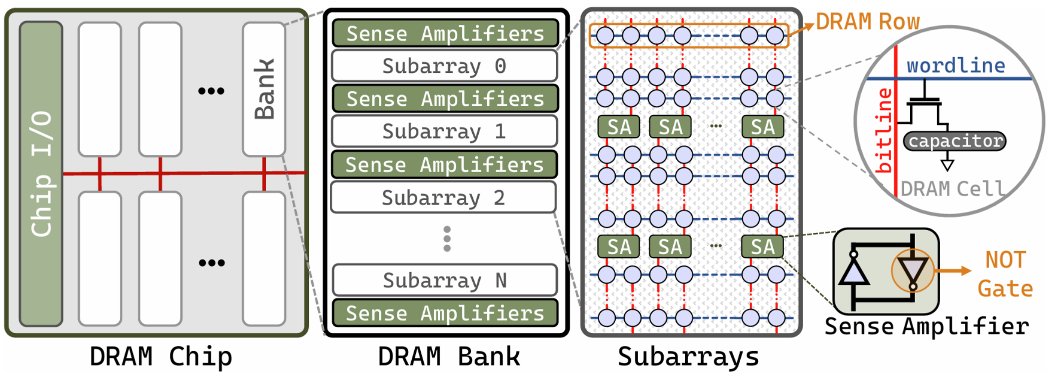

Fig. 2. Hierarchical organization of a modern DRAM system. The hierarchy spans from chips down to banks, subarrays, and transistors.[298]

interface width (typically 64 bits). Within a rank, parallelism is further exposed through **Banks**. Modern standards (e.g., DDR4 and DDR5) introduce **Bank Groups** to cluster banks, thereby reducing global bus contention and enabling higher burst data rates. The fundamental building block is the **Bank**, which operates independently to process commands. Internally, a bank is divided into multiple **Subarrays**, sharing global peripheral circuits such as row decoders and sense amplifiers.

DRAM chips are characterized by their **I/O width**, typically denoted as x4, x8, or x16 in DDR-style architectures. This notation specifies the number of data bits each chip contributes to the rank. For example, sixteen x4 chips or eight x8 chips are required to compose a standard 64-bit rank. While the hierarchical principles remain consistent, specific implementations vary across **technology standards** tailored for different domains [253]. The *DDR* series serves general-purpose computing; *LPDDR* targets mobile systems with low-power optimizations; *GDDR* offers wide interfaces for graphics workloads; and *HBM/HMC* leverage 3D-stacking and Through-Silicon Vias (TSVs) to deliver extreme bandwidth for high-performance computing.

Data access within this hierarchy is governed by strict **timing constraints** and row-level granularity. Although CPU requests are typically cache-line sized (e.g., 64 bytes), DRAM operations manipulate an entire row (typically 8 KB). The latency of these operations is dictated by key timing parameters: t_{RCD} (Row Address to Column Address Delay), t_{CL} (CAS Latency), and t_{RP} (Row Precharge Time). Collectively, these parameters define the minimum intervals between Activate, Read/Write, and Precharge commands. Understanding these constraints is critical for PIM architectures, as many designs aim to hide these latencies or exploit the internal bandwidth available within the row buffer before data traverses the hierarchy.

Managing this complex hierarchy is the responsibility of the **Memory Controller (MC)** (as shown in Figure 3). The MC serves as the bridge between the host processor and the DRAM subsystem, translating high-level memory requests into precise sequences of DRAM commands while strictly enforcing timing parameters (t_{RCD}, t_{RP}, etc.). Beyond basic scheduling, modern controllers implement sophisticated policies such as FR-FCFS (First-Ready, First-Come-First-Serve) to maximize row buffer hits and manage refresh operations. In the context of PIM, the memory controller often requires modification to support new instructions or to arbitrate between standard memory accesses and offloaded computation tasks.

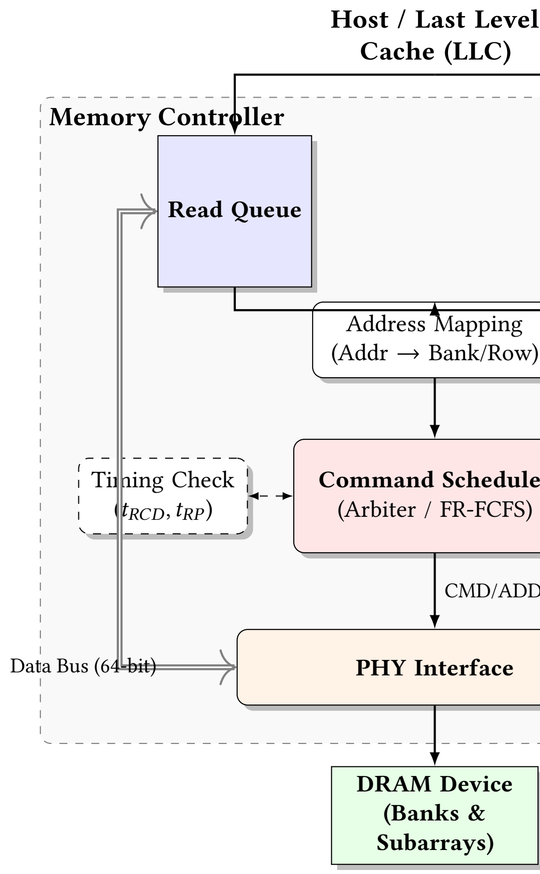

Fig. 3. Functional block diagram of a typical DRAM Memory Controller. It buffers requests in queues, translates addresses, and employs a scheduler to arbitrate commands based on timing constraints (t_{RCD}, t_{RP}) and refresh requirements, before issuing them to the DRAM via the PHY interface.

2.2 Representative Workloads

2.2.1 The Memory Wall in von Neumann Architectures. Traditional von Neumann architectures are fundamentally constrained by the physical separation of processing units and memory storage. While modern processors have achieved orders-of-magnitude improvements in computational throughput, the bandwidth and latency of the memory subsystem have failed to scale commensurately. This widening performance gap, known as the **Memory Wall**, results in systems where performance is increasingly dictated by the cost of data movement rather than arithmetic speed. This bottleneck is particularly severe for emerging data-intensive workloads, which can be broadly categorized into two types based on their memory access characteristics: those dominated by **massive data movement** and those exhibiting **irregular access patterns**.

The first category includes workloads such as Deep Neural Networks (DNNs), Large Language Models (LLMs), and Fully Homomorphic Encryption (FHE). These applications operate on massive datasets and parameters, requiring continuous high-bandwidth data streaming between memory and the processor. For instance, training and inferencing LLMs involve repeated matrix multiplications over gigabytes to terabytes of parameters, causing the execution time and energy consumption to be dominated by off-chip data transfer. In these scenarios, the system is strictly **bandwidth-bound**, as the memory channel cannot supply data fast enough to keep the compute units busy.

The second category comprises workloads characterized by sparse and irregular memory accesses, such as graph analytics, Graph Neural Networks (GNNs), and large-scale recommendation systems. Unlike the sequential streaming in dense DNNs, these applications often perform gather-scatter operations or pointer chasing (e.g., traversing edges in a graph or looking up embedding tables). Such patterns exhibit poor spatial locality, leading to frequent cache misses and low utilization of the memory burst bandwidth. Consequently, these workloads become **latency-bound**, where the processor spends significant cycles stalling for data to be fetched from DRAM. Both categories of workloads provide strong motivation for PIM architectures, which aim to either amplify internal bandwidth or reduce effective access latency by processing data in situ.

2.3 Processing-in-Memory Paradigms

To fundamentally mitigate the bandwidth limitations and access latency of the Memory Wall discussed in Section 2.2, PIM architectures advocate shifting the computation paradigm from processor-centric to memory-centric. By placing processing capabilities in close proximity to data storage, PIM architectures minimize data movement, thereby improving both system throughput and energy efficiency.

2.3.1 General Taxonomy: CIM vs. PNM. Broadly, PIM architectures can be classified into two primary categories based on the physical proximity of computation to the memory bitcells: **Compute-in-Memory (CIM)** and **Processing-near-Memory (PNM)**.

Compute-in-Memory (CIM), also referred to as *in-situ* computing, performs operations directly within the memory arrays. This category encompasses diverse technologies and computing mechanisms. For instance, **Analog CIM** typically exploits the intrinsic physical properties of Non-Volatile Memory (NVM) devices (e.g., RRAM, PCM) to perform parallel multiply–accumulate (MAC) operations in the analog domain, leveraging Ohm's and Kirchhoff's laws. **Digital CIM**, on the other hand, often utilizes SRAM or modified NVM arrays to embed logic gates (e.g., AND, OR, XOR) directly into the bitcell structures. These *in-situ* approaches offer high parallelism and density by tightly coupling storage and computation.

In contrast, **Processing-near-Memory (PNM)** retains the traditional digital abstraction but moves the processing units, ranging from simple arithmetic logic to general-purpose cores, physically closer to the memory arrays. Instead of modifying the bitcell itself, PNM integrates logic at the periphery of memory banks, within the buffer chips of memory modules, or in the base logic layer of 3D-stacked memories. This approach reduces the distance data must travel while utilizing standard memory interfaces and protocols.

2.3.2 Focus of this Survey: DRAM-Based PIM. Within this broad taxonomy, this survey focuses specifically on **DRAM-based PIM architectures**. DRAM-based solutions leverage the high storage density, cost-effectiveness, and mature manufacturing ecosystem of commodity DRAM. These architectures span the PIM spectrum: they include near-memory designs that integrate logic layers (enabled by emerging 3D-IC and hybrid bonding technologies) and **Processing-using-DRAM (PuD)** techniques that exploit internal circuit behaviors (e.g., RowClone, DRAM-LUT) to perform bitwise operations *in-situ*.

We focus on DRAM-based PIM not only due to its academic significance but also its practical feasibility and rapid industrial adoption. Notably, major memory vendors have recently demonstrated commercial prototypes, such as Samsung's HBM-PIM [158] and SK Hynix's GDDR6-AiM [166], validating DRAM-PIM as a viable solution for large-scale memory-intensive workloads.

3 Core Implementation and Classic Architecture

This section surveys the landscape of DRAM PIM hardware, categorizing different approaches based on the physical proximity of the computing units to the DRAM memory cells. This placement is a critical design choice that dictates the trade-offs between computational capability, available bandwidth, parallelism, and hardware overhead. We classify the primary implementation strategies into four main groups: Processing-using-DRAM, Processing-near-Bank, Rank-level Processing, and 3D Hybrid-bonding-based architectures.

3.1 Processing-using-DRAM: Real In-situ Parallelism

Processing-using-DRAM exploits the intrinsic analog behavior of DRAM subarrays—in particular, sense amplifiers and bitline charge sharing—to perform *bulk bitwise* operations directly inside the memory array. By computing where the data reside, these mechanisms aim to minimize off-chip data movement and expose massive internal bandwidth. To realize this in commodity *1T1C* cells, the peripheral circuits are modified to support multi-row activation (e.g., simultaneous or staged activation of multiple rows). This activation triggers charge sharing along the bitlines, driving sense amplifiers to realize a functionally complete set of Boolean primitives (*AND*, *OR*, *NOT*).

Ambit [242] pioneered this approach by enabling Triple Row Activation (TRA)(shown in Figure 4 directly inside commodity subarrays without major structural changes. It achieves up to 44.9× higher throughput and **35× lower energy consumption** for bitwise operations compared to an Intel Skylake processor. Beyond standard 1T1C designs, other prototypes like *DRISA* [176] adopt enhanced cell structures (e.g., *3T1C*) to provide finer-grained control over activation paths, organizing subarrays into reconfigurable parallel fabrics for richer logic support.

However, because these operations rely on bitline interaction, operands must strictly reside within the same subarray. Consequently, in-DRAM data movement primitives, such as *RowClone* (which copies rows via back-to-back Activate/Precharge), are essential to co-locate operands before computation, ensuring that array-level operations can be issued with minimal latency.

The defining strength of Processing-using-DRAM lies in its ability to expose the **massive internal bandwidth** of subarrays, enabling simultaneous operations on thousands of bits. This in-situ execution drastically **reduces off-chip data movement**, translating to significant performance and energy gains for data-intensive workloads, all while incurring **minimal area overhead** by repurposing existing structures rather than adding large logic units.

However, this approach faces distinct limitations impeding industrial adoption. First, functional completeness relies on composing complex arithmetic (e.g., multiplication) from basic bitwise primitives, incurring high latency and serialization costs. Second, the reliance on analog charge sharing makes these architectures highly sensitive to process variation, requiring substantial modifications to **DRAM timing, peripheral circuits, and control logic** to ensure reliability. Finally, the mandatory data alignment via *RowClone* adds overhead for irregular data patterns.

3.2 Processing-near-Bank: Exploring Bank-Level Parallelism

3.2.1 Core Idea. As shown in Figure 5, **Processing-near-Bank** architectures integrate lightweight processing units (PUs)—either small general-purpose cores or application-specific functional units (FUs)—into the peripheral logic of each DRAM bank, physically close to the sense amplifiers. The primary goal is to exploit the high **internal bank-level bandwidth**—typically an order of magnitude greater than TSV-limited data paths in von Neumann architectures [281]—thereby drastically reducing off-chip data movement. The choice of PU is strictly workload-oriented. For general-purpose offloading, designers may adopt tiny in-order CPUs (e.g., RISC-V) with minimal instruction storage; for high-throughput kernels, specialized FUs (e.g., **MAC arrays** for DNNs [166],

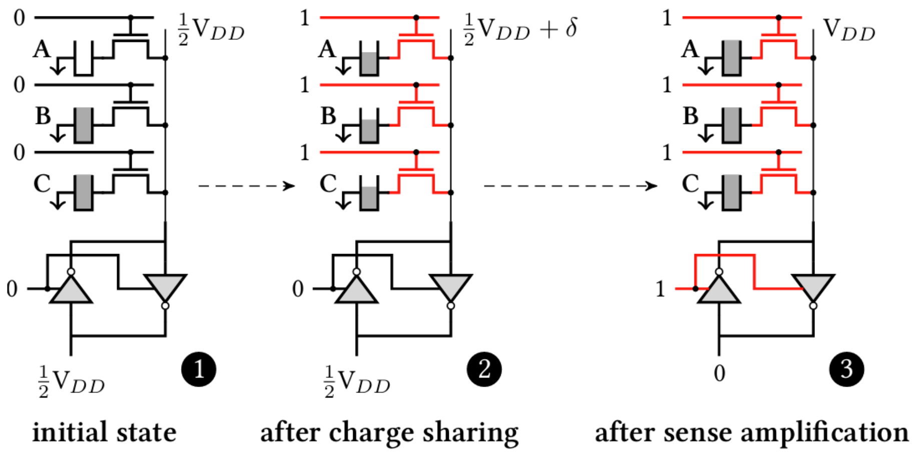

Fig. 4. Triple Row Activation-processing using DRAM

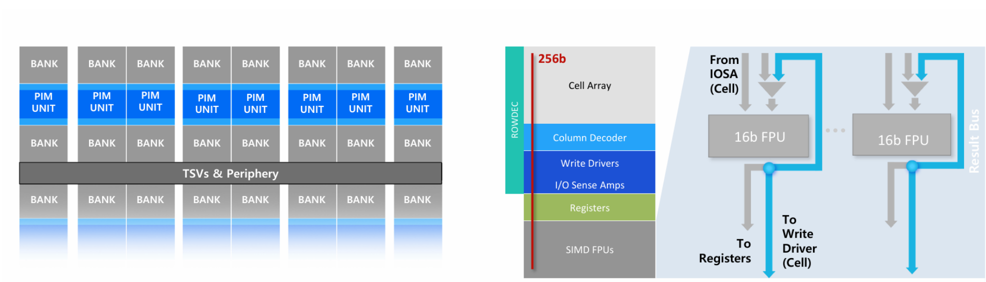

Fig. 5. Architecture of Processing-near-Bank designs, illustrating PUs integrated within bank peripheries to exploit internal bandwidth.

or **Hamming-distance/bitwise-popcount engines** for recommendation and search [179]) often deliver superior energy efficiency. Specifically, recent work [166] targets memory-bound deep-learning workloads such as RNNs and MLPs by integrating lightweight MAC-based PUs near each DRAM bank in a GDDR6-based Accelerator-in-Memory (AiM), achieving up to **1 TFLOPS peak throughput** and up to **10× system-level speedup** over a GPU+HBM2 baseline. To support execution, a small on-die storage (e.g., SRAM buffer [61] or PU-local register files [158]) holds operands, while a command/queue interface orchestrates data movement between the host and PUs.

3.2.2 Characteristics. This architectural approach offers distinct strengths: it balances **parallelism** (across many banks) and **compute density** (lightweight PUs), and leverages high internal row-buffer bandwidth with **minimal intrusion** into the sensitive cell array. Consequently, it effectively reduces off-chip traffic and can accelerate bandwidth-bound kernels such as **reductions, stencil/bitwise operations, and GEMV**. **However**, implementation faces notable constraints. First, per-PU compute capability is strictly limited by **area, power, and thermal budgets**, as well as

Table 1. Comparison between near-bank and rank-/DIMM-level PIM architectures.

Aspect	Near-bank PIM	Rank-/DIMM-level PIM
Bandwidth (local access)	~hundreds of GB/s per bank (internal)	~tens of GB/s (external channel limited)
Compute capability per unit	Low (lightweight PU, few GFLOPS)	High (FPGA/ASIC/CPU-level, tens–hundreds GFLOPS)
Parallelism scale	Fine-grained (tens–hundreds of banks)	Coarse-grained (a few PEs per module)
Accessible data range	Intra-bank or near-bank rows only	Full rank / DIMM address space

instruction and storage capacity. Second, **data locality** remains critical; moving data across subarrays or banks can dominate latency and energy unless aided by efficient copy or bridge primitives. Third, **cross-bank communication** bandwidth is constrained without dedicated on-stack networks, although recent research proposes on-stack bridging (e.g., NDPBridge [262]) to mitigate this overhead. Fourth, area and power overheads in the peripheral logic introduce inevitable **capacity and thermal trade-offs**. Finally, the **software stack complexity**—particularly regarding data placement, command scheduling, and cache coherence/visibility—remains non-trivial.

3.2.3 Representative Architectures. Representative near-bank architectures can be categorized by the underlying DRAM technology, each adopting different trade-offs. In the DDR domain, **UPMEM** [88] is a commercial PIM that integrates one tiny in-order core ("DPU") per bank in the peripheral logic. Each DPU owns its bank's **MRAM** and possesses small **IMEM/WRAM scratchpads**; programs use an SPMD style with explicit DMA between MRAM and WRAM, allowing systems to scale to thousands of DPUs across multiple DIMMs. For high-bandwidth scenarios, **HBM-PIM** [145] integrates lightweight programmable PCUs into each HBM2/2E bank, leveraging intra-bank bandwidth while maintaining full compatibility with standard interfaces. Results demonstrate over **2× performance improvement** and **70% energy reduction** on AI workloads as a drop-in replacement. More recently, GDDR-based designs like **PIM Is All You Need** [93] couple near-bank units with CXL-attached memory expanders to build a GPU-free inference platform for Large Language Models (LLMs). This architecture leverages the high internal bank-level bandwidth of GDDR6 and host–memory coherence via CXL to achieve competitive throughput and energy efficiency compared to GPU-based systems.

3.3 Rank-Level and DIMM-Level Processing: Compatible Integration

3.3.1 Core Idea. As summarized in Table 1 and illustrated in Figure 6, **Rank- or DIMM-level PIM** adopts a different integration strategy compared to near-bank designs. Instead of embedding logic deep within the memory arrays, this approach integrates one or several powerful **Processing Elements (PEs)**—such as FPGA fabrics, small general-purpose CPUs, or custom ASIC accelerators—directly onto the memory module, typically within or adjacent to the buffer chip (e.g., RCD or DB) that manages the DRAM ranks. This placement introduces a fundamental trade-off: it sacrifices the fine-grained parallelism and massive internal bandwidth of bank-level integration in exchange for **higher compute capability** and **wider address visibility**. Physically, the PEs are one hop away from the cell arrays through the module interface, resulting in lower effective bandwidth and higher access latency; however, the relaxed area and thermal constraints at the module level allow for significantly more powerful logic units. A key advantage of this paradigm

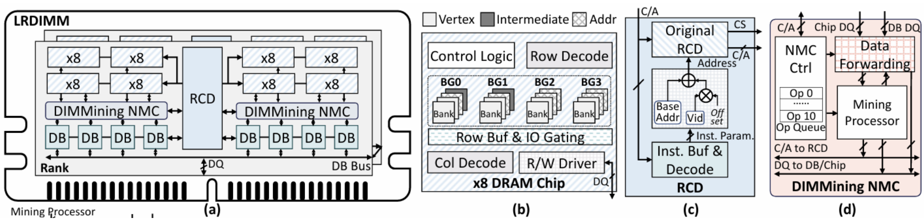

Fig. 6. Architecture of DIMM-level PIM, showing processing units integrated into the buffer chip or module PCB.

is **compatibility**: because modifications are confined to the module PCB or buffer chips, these architectures effectively support **commodity DRAM chips** and standard interfaces, simplifying prototyping and deployment as plug-in accelerators.

3.3.2 Characteristics. The primary strength of rank-level processing lies in its **flexibility and ease of deployment**. By avoiding modifications to sensitive DRAM dies, these designs function as drop-in replacements compatible with existing standards (e.g., DDR4/5, HBM2E) while supporting rich instruction sets and standard toolchains (e.g., Linux or FPGA runtimes). Furthermore, the larger power budget enables **higher compute capability per unit**, allowing for complex vector units or specialized accelerators, and supports **system-level scalability** by plugging additional modules into empty channels. **However**, this approach faces inherent constraints. The most critical is the **bandwidth limitation**, as PEs are constrained by the external rank-level channel (tens of GB/s) rather than the multi-hundred-GB/s internal bandwidth available to near-bank PIM. Additionally, data must traverse the I/O interface between DRAM chips and the buffer, creating an **energy efficiency gap** compared to in-DRAM computation. Other challenges include the **thermal density** of placing high-power logic on compact modules and the **granularity mismatch** where fewer, coarse-grained units struggle to exploit massive data parallelism.

3.3.3 Representative Architectures. We highlight representative designs from industry and academia that illustrate the diversity of this approach. **AxDIMM (Samsung Prototype)** [133, 161] is a prominent industrial prototype that embeds FPGA-based units into the buffer chip of standard DDR4 DIMMs. It accelerates database scans and recommendation inference by exploiting intra-module bandwidth, achieving **6.8× higher throughput** for scans and **2× energy efficiency** for recommendation tasks compared to CPU baselines. **RecNMP** [132] focuses specifically on memory-bound sparse embedding operations. By placing lightweight specialized logic in the buffer chip and employing optimizations like table-aware scheduling, it achieves a **9.8× latency speedup** and **45.8% energy savings** for recommendation models. **TensorDIMM** [156] targets tensor workloads by integrating a systolic accelerator connected to DRAM ranks via standard interfaces, delivering an order-of-magnitude higher energy efficiency for GEMM kernels. Similarly, generic prototypes like **DIMMing** [54] explore integrating RISC-V elements to support diverse data-parallel workloads via software-controlled offloading. Finally, **Pyramid** [324] proposes a *Processing-in-Hierarchical-Memory (PiHM)* architecture for billion-scale Approximate Nearest Neighbor (ANN) search. It coordinates distributed rank-level units for fine-grained graph traversal with in-storage computing for coarse-grained scanning, achieving **50× higher throughput** than CPU/GPU systems by mapping computation to the optimal memory tier.

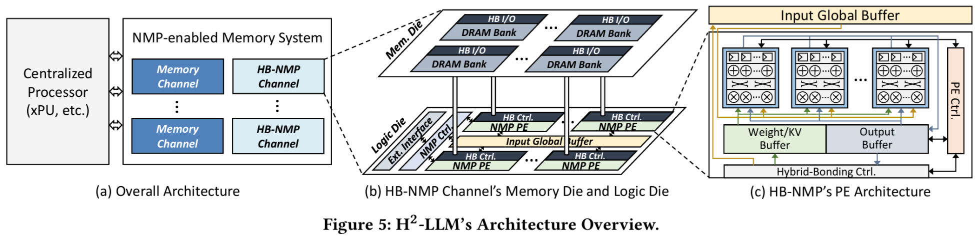

Fig. 7. Architecture of 3D Hybrid-Bonding PIM, illustrating the logic die stacked beneath DRAM layers (Source: H2LLM).

3.4 3D Hybrid-Bonding-Based PIM: The High-Bandwidth Frontier

3.4.1 Core Idea. As shown in Figure 7, compared with conventional in-die or near-bank PIM designs, **3D hybrid-bonding-based PIM (HB-PIM)** enables significantly higher memory–logic bandwidth by vertically stacking a dedicated logic die beneath one or more DRAM dies. Instead of integrating computing units within the DRAM array itself, HB-PIM relocates the logic circuits to a separate but tightly coupled die, allowing both **greater transistor density** and **higher power budgets**. The dies are interconnected through **ultra-dense vertical links** that combine through-silicon vias (TSVs) with **Cu–Cu hybrid bonding technology**, which offers interconnect densities exceeding ten thousand connections per mm^2. This configuration transforms the DRAM–logic interface from a conventional off-chip I/O boundary into a fine-grained, on-die-level network. Such a stacked organization effectively forms a tightly integrated **compute cube** in which computation and data are colocated in three dimensions. This enables massive vertical bandwidth (potentially several hundreds of GB/s per stack) and extremely low communication latency between compute and memory layers.

Notably, the concept of 3D hybrid-bonding PIM does not conflict with other paradigms such as near-bank PIM or subarray-level PIM. Rather, these approaches can coexist hierarchically: coarse-grained data-parallel operations can be offloaded to near-bank PIM, while fine-grained or control-intensive tasks are handled by the logic layer in the 3D stack. The emergence of hybrid bonding as a manufacturable technology has also inspired active efforts in **software-hardware co-design**. Toolchains, compiler extensions, and runtime systems are being developed to expose the compute capabilities of HB-PIM to programmers, similar to how GPU programming models evolved in the past decade.

Despite its potential, HB-PIM still faces major technical barriers. **Thermal management** is particularly challenging because the high-power logic die is buried underneath multiple DRAM layers, impeding heat dissipation. In addition, manufacturing complexity, yield degradation, and alignment precision in hybrid bonding remain critical bottlenecks; addressing these challenges will be key to realizing practical large-scale deployment. A unique architectural challenge is **NUMA-like Latency Heterogeneity**, which primarily arises in logic-die-based HB-PIM architectures due to non-uniform communication costs within the 3D stack. Since the computing logic resides on one die while multiple memory dies are stacked vertically, accessing the memory die immediately adjacent to the logic die is fast, but accessing a die further up the stack incurs higher latency due to longer vertical signal paths. This creates a NUMA-like effect where different DRAM layers have distinct access latencies from the same processing unit, requiring intelligent data placement and scheduling.

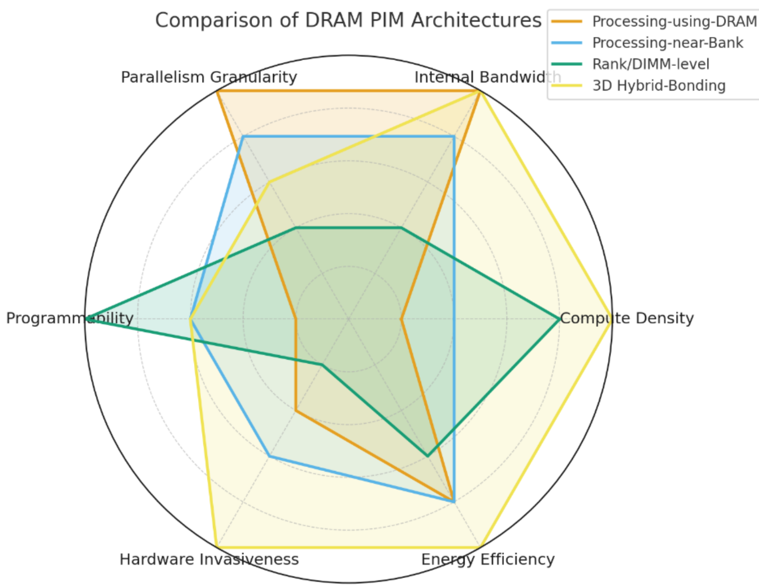

Fig. 8. Qualitative comparison of DRAM-PIM architectures across key metrics (computational granularity, parallelism, bandwidth, programmability, and hardware invasiveness).

3.4.2 Characteristics. HB-PIM provides **unprecedented logic–memory coupling** with near-zero data movement overhead. It enables general-purpose or domain-specific accelerators to operate directly within the memory stack, delivering orders-of-magnitude improvements in effective memory bandwidth and energy efficiency for data-intensive workloads such as **AI inference, graph analytics, and in-memory databases**. The tight vertical integration also opens opportunities for new architectures such as **memory-centric chiplets** and reconfigurable PIM fabrics. **However**, the main drawbacks lie in high fabrication cost, limited scalability, and severe thermal constraints. The logic die's power dissipation is trapped beneath DRAM layers, making conventional cooling solutions ineffective. Moreover, testing and yield management become substantially more complex due to the increased number of bonding interfaces. Finally, the lack of a standardized programming model and toolchain still hinders software ecosystem maturity.

3.4.3 Representative Architectures. **3D-PATH** [296] is a 3D hybrid-bonding accelerator for LUT-based PIM that proposes a hierarchical LUT design (a fast LUT on the logic die and a large LUT on the memory die). This co-design optimizes storage and computation, achieving up to **12.68× higher throughput** over GPUs. **Stratum** [216] provides a co-design framework for MoE LLM serving on monolithic 3D DRAM. It introduces **in-memory tiering** to map hot/cold experts to fast/slow memory layers, exploiting latency variations to achieve up to **8.29× higher throughput** than GPUs. **H2-LLM** [171] presents a heterogeneous hybrid-bonding accelerator for edge LLM inference featuring a design space exploration (DSE) framework to co-optimize hardware and dataflow. The resulting design achieves a **2.72× geomean speedup** over in-die NMP architectures.

3.5 Summary and Comparison

As illustrated in Figure 8, we present a qualitative comparison of the four approaches across key metrics: **computational granularity**, **parallelism**, **bandwidth**, **programmability**, and **hardware**

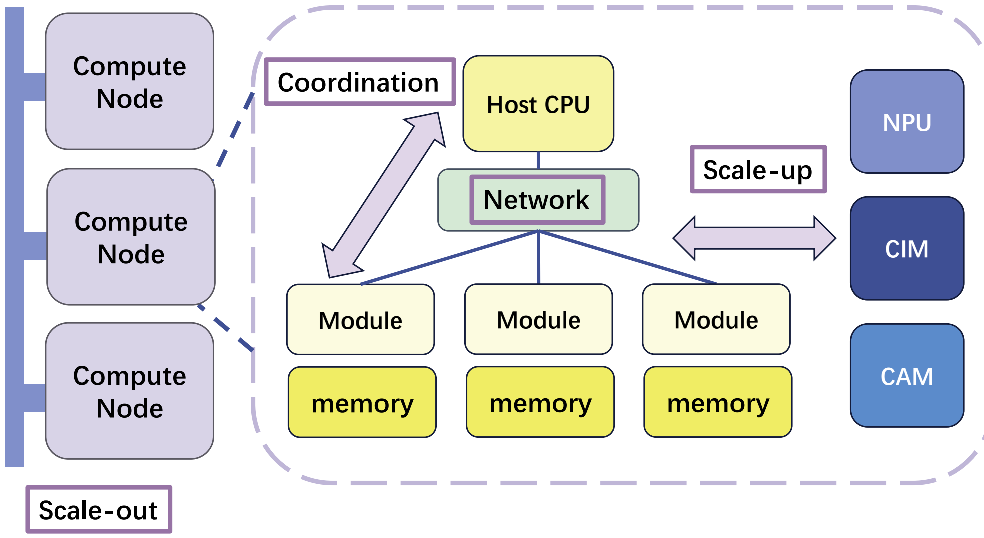

Fig. 9. System Extensions: Communication, Coordination, Scale-up and Scale-out

invasiveness. **Processing-using-DRAM** exposes the largest **in-situ parallelism** and near-zero array–periphery data movement, but its **Boolean–primitive granularity**, operand colocation constraints, and reliability concerns limit general-purpose applicability. **Processing-near-bank** strikes a pragmatic balance: lightweight PUs at bank peripheries harvest high **internal row-buffer bandwidth** with modest hardware intrusion, yet still face cross-bank movement and software orchestration overheads. **Rank-/DIMM-level designs** trade internal bandwidth for **stronger, more programmable PEs** and commodity compatibility, making them attractive for coarse-grained kernels but inherently **channel-limited**. Finally, **3D hybrid-bonding architectures** push the **bandwidth–density frontier** by tightly coupling DRAM stacks with a logic die, enabling rich accelerators but facing **thermal**, **yield**, and **NUMA-like latency heterogeneity** inside the stack.

Across all classes, the governing axes defining the design space are **proximity to cells**, **effective bandwidth per byte moved**, **compute density per area/power**, **address visibility**, and **programmability**. Observing the evolution of these designs reveals a clear trend from specialized, high-overhead mechanisms toward more general-purpose, modular approaches enabled by advanced packaging. Consequently, the future trajectory points toward **heterogeneous, hierarchical PIM** systems. Such systems will rely on unified runtimes and dataflow-aware scheduling to place the right kernel at the right memory tier, co-optimizing data placement, movement, and parallelism for end-to-end efficiency.

4 System-Level Optimizations for PIM Integration

This section addresses the key challenges and solutions related to inter-PIM communication, host-PIM coordination, and overall system scalability. The research discussed here aims to build a synergistic and expandable PIM ecosystem, moving beyond the capabilities of individual PIM devices.

4.1 Inter-PIM Communication

4.1.1 The Problem: The Host-Mediation Bottleneck. In many PIM systems, communication between distinct PIM units (e.g., across different banks, chips, or DIMMs) is typically mediated by the host CPU. Consequently, data must be transferred from the source PIM unit to the host's caches and then written back to the destination PIM unit. This host-mediated forwarding creates a severe performance and energy bottleneck. It consumes valuable host CPU cycles, pollutes caches, and serializes traffic through the relatively narrow host-memory bus, negating the very purpose of near-data processing.

4.1.2 Solutions: Controlling Traffic and Increasing Bandwidth. **Software-based approaches** aim to improve inter-PIM communication through system-level abstraction and runtime coordination, offering the advantage of being readily deployable on existing hardware. They enhance programmability and portability by orchestrating data movement and synchronization among distributed PIM units without requiring new interfaces or physical links. For example, **SimplePIM** provides high-level APIs that expose communication primitives among PIM cores and between PIM and the host, simplifying coordination in real PIM systems [34]. **PID-Comm** further models PIM processing elements as a multi-dimensional topology and optimizes collective patterns such as reduce and broadcast, achieving up to 4.2× performance improvement over baseline implementations [210]. However, these approaches remain constrained by host-managed memory channels and cannot fundamentally eliminate the traffic serialization through the CPU–memory interface.

To overcome this inherent limitation, **hardware-based approaches** (targeting rank, DIMM, and bank levels) introduce hardware-assisted interconnects that enable *direct* PIM-to-PIM data exchange. At the DIMM or rank level, **DIMM-Link** establishes lightweight links between DIMMs to allow cross-DIMM communication without host intervention [322]. At the bank level, **NDPBridge** adds on-chip "bridges" to connect near-bank compute units across the DRAM hierarchy, thereby reducing latency and energy relative to host forwarding [262]. Building on these ideas, **PIMnet** designs a multi-tier, domain-specific interconnection fabric aligned with the DRAM hierarchy. Its ring-based inter-bank topology and hierarchical scheduling eliminate dynamic routing overhead, enabling up to 85× speedup for collective operations and 11.8× on real applications compared with baseline PIM systems [252].

4.2 Host CPU-PIM Coordination: Scheduling for computation

4.2.1 The Problem: Serial Execution and Memory-Space Contention. Following the previous section on *communication*, this part focuses on challenges in the *computation* phase of PIM systems. Unlike distributed-memory architectures where each node operates independently, PIM systems always involve a host CPU that not only issues commands and controls execution flow but may also participate in computation. This host–PIM relationship introduces two major inefficiencies: One primary issue is **Serial Coordination and System Stalls**,where coordination between the host processor and PIM accelerators is often coarse-grained and sequential. When PIM kernels occupy the memory channels, the host CPU is forced to wait, leading to substantial idle cycles. Conversely, when the CPU dominates memory bandwidth, PIM units remain stalled. Such serialized execution prevents concurrent utilization of compute and memory resources, resulting in overall underutilization of the system. In addition to execution stalls, systems also suffer from **Memory-Space Contention**. This arises because the host CPU and PIM units exhibit fundamentally different memory access behaviors: CPUs favor fine-grained interleaving to maximize bandwidth, while PIMs prefer large contiguous data regions to exploit internal parallelism. When both share the same physical memory space without coordination, their conflicting access patterns cause interference, bandwidth contention, and unnecessary data movement across CPU–PIM boundaries.

4.2.2 Solutions: Overlapping Compute and Memory-Space Coordination. Recent studies approach this problem from two complementary perspectives: (1) increasing parallelism to better overlap computation and data transfer, and (2) coordinating memory space usage between the host CPU and PIM units to mitigate contention.

The strategy of **Increasing Parallelism** seeks to reduce idle periods by enabling the host (or CPU) and PIM units to operate concurrently, rather than strictly sequentially. For example, **OverlaPIM** applies dependency-aware mapping so that a subsequent DNN layer begins execution before the prior layer has fully completed, achieving $\sim 2.1\times$–$4.1\times$ speedup [317]. Similarly, **HAIL-DIMM** interleaves host and near-data accesses at the bank level, enabling fine-grained concurrency between CPU requests and PIM operations on the same memory channels [164].

On the other hand, **Memory-Space Coordination** focuses on aligning the contrasting memory-access patterns of host CPUs (which prefer finely interleaved, multi-channel reads) and PIM units (which thrive on large contiguous chunk accesses). To this end, **UM-PIM** introduces a unified and shared virtual memory space for CPU and PIM, eliminating explicit data copies and reducing access conflicts between interleaved and chunked access patterns [310]. Additionally, **PIM-Tree** proposes a skew-resistant in-memory index that dynamically divides workloads between host CPUs and PIM nodes through a push/pull mechanism, balancing load and improving performance under skewed queries [129].

4.3 Deep Integration into Heterogeneous Systems

Given that PIM is compute power-limited as shown in Figure 10, its future is not as a standalone replacement for CPUs or GPUs, but as a synergistic component within a larger heterogeneous ecosystem. True integration will occur at multiple layers of the system hierarchy, from device-level packaging to runtime software and application scheduling. Designing such hybrid systems requires considering a wide range of factors, including the computing capability of each device, the balance between memory access and computation intensity (e.g., operational intensity as one indicator), the execution characteristics unique to each architecture, and the communication topology and bandwidth among components. No single metric can capture this complex design space; rather, the challenge is to understand how these factors interact to determine overall system efficiency.

Addressing **intra-node heterogeneity** (CPU–NPU–PIM Synergy) requires intelligently partitioning workloads within a node to leverage the distinct strengths of each component. As new applications (e.g., MoE, RAG) and new PIM technologies (e.g., 3D hybrid-bonded PIM) emerge, system designers must revisit how to balance computing, bandwidth, and capacity to achieve efficiency. Integrating diverse devices brings flexibility but also introduces a high-dimensional design space—encompassing mapping, scheduling, memory partitioning or sharing, capacity allocation, power distribution, and temporal variation in computation patterns. Purely heuristic approaches offer fast adaptation but rarely achieve global optimality; full design-space exploration is thorough but expensive. Practical systems should combine both, guided by analytic models, heuristics, and empirical feedback.

In terms of **workload partitioning and architectural heterogeneity**, a common principle is to assign tasks based on their computational and memory characteristics—offloading data-intensive, memory-bound operations to PIM, while compute-intensive operations remain on GPUs or CPUs. Yet real-world systems demand more nuanced partitioning that also considers data locality, interconnect cost, and concurrency. **Duplex**[300] illustrates this synergy: by combining GPU cores with logic-layer PIM on the same HBM stack, it dynamically co-processes different sub-stages of large-model inference to balance throughput and energy efficiency. **PAPI**[101] extends this idea further, employing multiple specialized PIM types—one optimized for computation (FC-PIM) and another for capacity and bandwidth (Attn-PIM)—together with GPUs to adapt to different phases

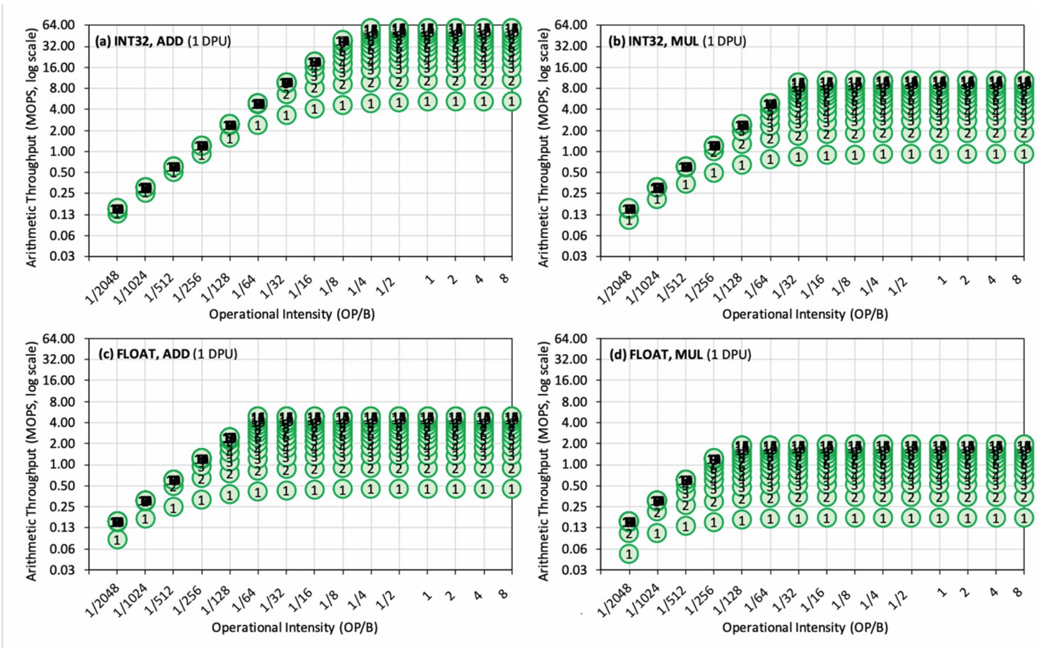

Fig. 10. Roofline anlysis for PIM: Compute power is limited. The number inside each dot indicates thenumberof tasklets.[88]

of LLM decoding. **Pyramid** [324] further shows a DRAM–SSD integrated PIM for graph-based ANNS at billion scale: a hierarchical graph-cluster ANNS reshapes access patterns (small-graph irregular in memory, in-cluster sequential in storage), with distributed distance computation and centralized sorting in DIMM and in-storage sequential processing on SSD, yielding order-of-magnitude throughput gains under the same recall.

Moving **beyond partitioning** to co-design opportunities, another underexplored but promising direction is to build upon PIM's intrinsic characteristics rather than treating it as an auxiliary accelerator. By identifying which operations are inherently efficient for near-memory execution, developers can reformulate algorithms to align with PIM's strengths—transforming high-level computation into forms natively supported by the memory substrate. This philosophy follows the broader principle of software–hardware co-design: abstracting applications into modular operations, mapping them to the most suitable compute units, and jointly optimizing data layout, scheduling, and synchronization. Achieving such holistic integration will ultimately require unified runtime support, standardized programming models, and quantitative models that balance compute, communication, and storage resources across heterogeneous nodes.

4.4 System Expansion: Multi-Node

4.4.1 Motivation and the Need for CXL. While earlier sections focus on scale-up within a single node, emerging workloads such as large language models and large-scale recommendation continue to exceed the memory capacity and bandwidth that even PIM-equipped systems can provide. A single node cannot scale beyond its limited memory channels, DIMM capacity, and power or thermal constraints. To sustain PIM's benefits at larger working-set sizes, systems must extend toward a disaggregated setting that offers coherent, high-capacity memory pools with low host overhead. Compute Express Link provides such a substrate by enabling cache-coherent load or store access to far memory devices and replacing the rigid multi-drop DRAM interface with a flexible switch-based topology.

4.4.2 Challenges When Extending PIM over CXL. Although CXL offers capacity and composability, several key challenges arise when deploying PIM or NDP over it. One class of challenges is communication related. Transitioning from the DIMM multi-drop interface to a CXL switch fabric reduces contention but introduces nontrivial start-up latency on CXL links. PIM workloads often consist of fine-grained operations, and repeatedly issuing these small requests leads to significant instruction traffic that can dominate the overall cost. Furthermore, as more distributed PIM or NDP units are attached, it becomes difficult to maintain load balance and to prevent cross-device collectives from saturating the fabric. If the host CPU must remain in the critical path for frequent global coordination, communication bottlenecks reappear, and algorithm-level restructuring may even be required.

A second class of challenges is computation related. Each CXL device remains constrained by strict power and area limits, restricting the amount of available near-data compute. Using CXL.io for NDP invocation exacerbates the issue because it is optimized for device management rather than frequent offload. Issuing many small kernels through CXL.io thus incurs high latency, poor concurrency, and undesirable control overhead. Together, these constraints highlight that new offload paths and execution models are required to make CXL-based PIM effective.

4.4.3 Representative Architectural Solutions: CXL as a Coherent PIM and NDP Fabric. Researchers have proposed architectural extensions that use CXL to build scalable near-data compute systems while directly addressing the communication and computation bottlenecks described above. M^2NDP[98] provides a general-purpose NDP architecture for CXL.mem that replaces high-latency CXL.io invocation with *memory-mapped functions*, enabling offload using CXL.mem read or write packets. This eliminates most host involvement and reduces start-up overhead. M^2NDP also introduces *memory-mapped microthreading*, allowing a collection of lightweight hardware threads inside each device to support concurrent fine-grained tasks despite limited compute resources. Together, these mechanisms reduce control overhead, improve concurrency, and make NDP practical in CXL-based systems.

CLAY[301]focuses on embedding layers and demonstrates how to restructure NDP for CXL disaggregation by moving away from the limitations of DRAM multi-drop buses. CLAY reorganizes NDP units into a CXL-based hierarchy that performs local reduction at each memory module and global reduction inside the memory system, significantly reducing traffic to the host CPU. It further applies fine-grained memory mapping to reduce load imbalance and uses hierarchical aggregation to limit cross-device communication. These design choices mitigate the communication overhead intrinsic to distributed PIM and improve scalability when embedding tables span many CXL-attached modules.

Finally, we note that most CXL-based PIM and NDP evaluations rely on simulators or pre-silicon prototypes. Significant system-level development remains necessary before rack-scale deployment, including runtime support, coherence management, security and freshness guarantees, and observability infrastructure.

4.5 Summary of System-Level Approaches

In summary, realizing the full potential of PIM requires moving beyond device-level innovations to holistic system-level optimizations. This section highlighted that alleviating the host-mediation bottleneck is critical, necessitating direct inter-PIM communication fabrics and concurrent host-PIM scheduling strategies to maximize resource utilization. Furthermore, we emphasized that PIM acts as a synergistic component within heterogeneous architectures, requiring sophisticated workload partitioning and co-design to balance compute-bound and memory-bound operations. Finally, CXL provides the necessary substrate for multi-node scalability, though effective disaggregation demands optimized offloading protocols to overcome communication and power constraints. Together, these advancements transition PIM from isolated accelerators to a scalable, integral pillar of the future computing hierarchy.

5 Software Stack Design / Design tool flow

Unlocking the full potential of DRAM-based Processing-in-Memory (PIM) requires addressing two fundamental questions: how to accurately **evaluate** system performance, and subsequently, how to **leverage** these insights to drive efficient design and utilization. The foundation of this software ecosystem lies in establishing robust evaluation metrics. Unlike traditional architectures, PIM designs involve complex interactions between compute intensity, memory bandwidth, and thermal constraints that cannot be captured by simple throughput numbers alone.

Therefore, the design flow begins with **Simulation and Evaluation Frameworks**, which provide the necessary visibility to define and measure these metrics. Guided by these performance indicators, we then move up the stack to **Programming Models and Compilers**, which use static metrics to optimize code generation and offloading decisions, and **Runtime Systems**, which rely on dynamic metrics for adaptive scheduling and data management. Finally, we discuss **Design Space Exploration (DSE)**, where these metrics close the feedback loop, enabling automated tools to traverse the vast hardware-software design space to identify optimal configurations. This section details this essential toolchain, tracing the path from defining metrics via simulation to realizing high-performance PIM systems through intelligent software and automated design.

5.1 Simulation and Evaluation Frameworks

5.1.1 Objective. Conventional DRAM simulators are insufficient to model Processing-in-Memory (PIM) systems, as they capture only memory access behavior while ignoring in-memory computation and data movement. Dedicated DRAM-PIM simulators are therefore required to accurately represent compute units within DRAM, host–PIM interactions, and the resulting timing and power characteristics.

The first step toward designing and utilizing PIM systems is identifying appropriate evaluation metrics, which heavily depend on reliable simulation frameworks. In this subsection, we focus on simulators designed for DRAM, PIM, or hybrid DRAM–PIM architectures.

Simulator design is an intrinsic trade-off among **accuracy**, **simulation speed**, and **generality**—that is, the ability to model diverse near-data processing architectures and heterogeneous workloads. According to their modeling granularity and runtime efficiency, simulators can be categorized, from slowest to fastest, as **FPGA-based emulators**, **cycle-level simulators**, **instruction-level simulators**, and **behavior-level simulators**. From the workload perspective, simulators can also be divided into those targeting AI/ML workloads and those supporting general-purpose applications. The role of a simulator is twofold: (1) to provide accurate performance and power estimations—though often at the cost of long runtime—and (2) to offer a flexible environment for designers to quickly validate and iterate on architectural ideas. Therefore, simulators must allow easy modification of hardware and software parameters and support the implementation of new scheduling or mapping policies at both the architecture and application levels. Overall, these tools enable **rapid prototyping**, **pre-silicon performance analysis**, and **software stack co-design** for emerging PIM systems.

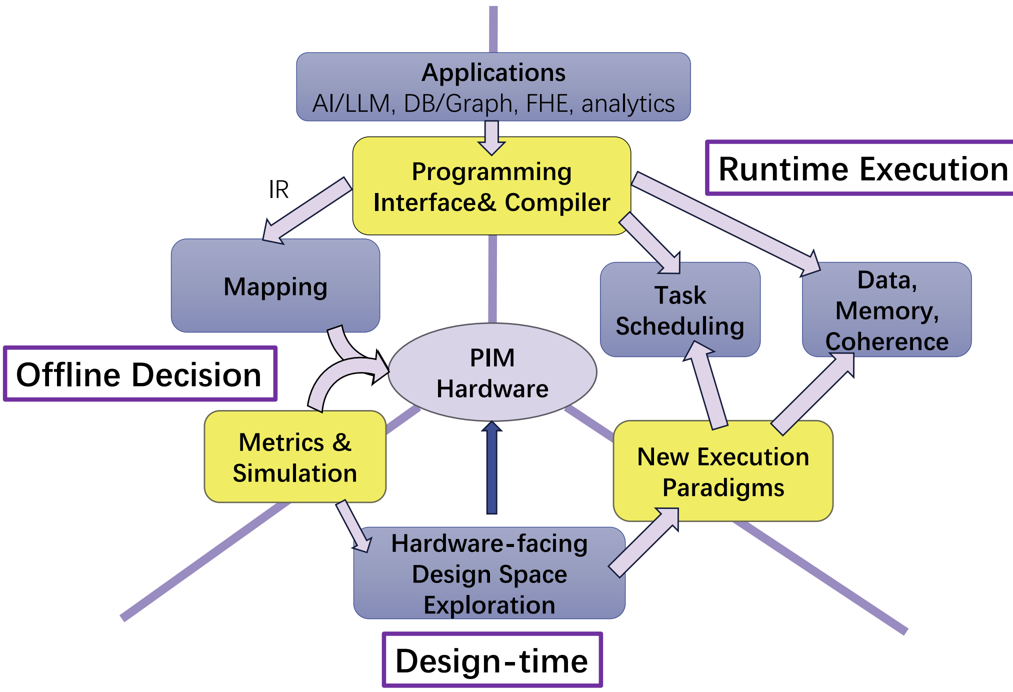

Fig. 11. Software tool

5.1.2 Types of Simulation Frameworks. Simulation frameworks for DRAM-based PIM research generally fall into two categories, reflecting the fundamental modeling priorities in the community: those emphasizing DRAM timing fidelity and those focusing on full-system execution. **DRAM-centric extensions**, such as *PIMSimulator* [232] (built upon *DRAMSim* [267]) or frameworks derived from *Ramulator* [151], extend DRAM timing models to incorporate in-memory compute units. Conversely, **System-level extensions** treat DRAM–PIM as a heterogeneous multi-core system. For instance, *PIM-Cloud* [42] adapts simulators like *zSim* [233], while other approaches integrate CPU and DRAM models—such as combining *Ramulator* with *zSim* or extending *gem5* [24]—to model PIM units as co-processors or memory-side devices.

Cycle-accurate simulators provide fine-grained microarchitectural visibility into compute units and memory controllers, enabling detailed performance and power analysis. However, they are extremely time-consuming, sometimes requiring several days to simulate modest workloads. To improve scalability, **higher-level simulators**—such as *UniNDP* [280] and *PIMEval* [247]—abstract away microarchitectural details, offering instruction-level or behavioral-level modeling. While this reduces accuracy, it allows general representations of diverse PIM architectures and significantly accelerates the exploration of design alternatives.

Regarding **FPGA-based emulators**, at the highest fidelity, they execute real applications on near cycle-accurate hardware models, providing critical system-level insights and validation. Representative examples include *PiMulator* [202] and *PRIMO* [103], which support full-stack evaluations from hardware logic to software runtime. Additionally, commercial DRAM-PIM devices such as *UPMEM* serve as de facto evaluation platforms, offering empirical observations of performance, programmability, and scalability.

5.1.3 Current Challenges and Future Directions. Despite the progress in simulation infrastructure, several critical challenges remain. **Host–PIM Co-Simulation** represents a significant gap, as most existing DRAM-PIM simulators emphasize memory-side computation while often oversimplifying or neglecting the host CPU's role. Future frameworks must therefore incorporate robust co-simulation of both the host and PIM sides to accurately model task partitioning, communication overheads, and cooperative execution strategies, such as those seen in *PIM-Tree*.

Closely related to modeling accuracy is the challenge of **Cross-Level Calibration**. Different abstraction levels are typically validated hierarchically—instruction-level simulators against cycle-level models, and cycle-level models against FPGA emulation or real hardware like *UPMEM*. However, frequent modifications to architectural parameters or internal structures make continuous manual calibration impractical, inevitably leading to accuracy loss. To address this, the community needs to adopt automated calibration pipelines or statistical approximation methods that can maintain fidelity across evolving designs.

From a usability standpoint, **Maintainability and Extensibility** pose additional barriers. Modifying hardware structures or scheduling mechanisms in current tools typically requires deep, error-prone changes to simulator internals. Future development should prioritize modularity, open interfaces, and scriptable configuration layers to facilitate rapid architectural exploration without the burden of complex refactoring.

Finally, the simulation ecosystem is constrained by the **Lack of Commodity PIM Platforms**. Unlike the GPU ecosystem, PIM lacks standardized, commodity-grade devices, leading to fragmentation that complicates validation. This situation raises fundamental open questions regarding why PIM architectures have not yet converged toward a common standard, which designs achieve the optimal balance between performance and programmability, and how the community can establish a unified software–hardware co-design ecosystem comparable to *CUDA* or *ROCm*. Addressing these issues is essential to transition PIM simulation frameworks from research prototypes to robust, industry-supported ecosystems.

5.2 Programming Models and Compilers

Programming for DRAM-PIM fundamentally differs from programming for traditional CPU systems. In CPU-centric architectures, computation largely abstracts away from data placement: memory accesses flow through a deep cache hierarchy, and compilers can treat most data-movement costs as uniform or amortizable. In DRAM-PIM, however, performance is dominated by *where* data resides and *how* it is accessed. Memory layout, access granularity, bank/subarray locality, and cross-device synchronization are all first-order concerns that directly shape execution efficiency. As a result, principled programming models and compiler support become indispensable for orchestrating compute–memory co-design, exposing the right levers to control data locality, and automating the mapping of applications onto heterogeneous CPU–PIM platforms.

To address this challenge, the overarching **objective** is to build an *accessible yet powerful* environment that maps applications onto heterogeneous CPU–PIM systems automatically, while still exposing expert controls when needed. Concretely, this design caters to two distinct user personas. For **customers (productivity-first)**, who prioritize ease of use without needing to understand microarchitectural details, the system provides a *minimal, friendly interface*—consisting of high-level task descriptions and few configuration choices—and relies on the compiler and runtime to choose a near-optimal mapping automatically. In contrast, for **experts (performance-first)** who demand maximum efficiency, the environment exposes *fine-grained control* over data placement, scheduling, synchronization, and CPU–PIM co-execution through composable policies and hints. Ultimately, the design must simultaneously balance *flexibility, generality, and performance*, cleanly separating the default automated path from optional expert overrides.

5.2.1 Programming Interfaces and Languages. The primary **goal** of PIM programming interfaces is to abstract away hardware complexity while exposing the *necessary* levers to control *data locality* and *where/when PIM executes*. This necessitates a fundamental **paradigm shift** from compute-centric to data-centric execution: instead of always moving data to a powerful compute unit, the system moves computation to the data's physical location, requiring the runtime and allocator to respect this intent.

To ground the discussion on real hardware, we categorize representative interfaces from UPMEM's ecosystem, spanning from low-level device management to higher-level parallel abstractions. As summarized in Table 2, these interfaces range from raw SDK primitives to sophisticated transactional runtimes:

- *UPMEM SDK (host-side).* A C-based runtime (with an optional Python host API) exposing explicit control of DPU resources, program loading, and data transfers between host memory and each DPU's MRAM. Developers manually manage the DPU set and data placement through primitives such as `dpu_alloc` (device allocation), `dpu_load` (kernel deployment), `dpu_copy_to`/`dpu_copy_from` (MRAM transfers), and `dpu_launch` (device-side execution). This layer represents the raw *host-side orchestration and data-movement model* for UPMEM systems.
- *UPMEM SDK (DPU-side).* Low-level primitives available within each DPU kernel, including `mem_alloc`, `mram_read`, `mram_write`, `barrier_wait`, and debugging utilities such as `dpu_printf`. These interfaces form the *on-device memory and synchronization substrate* that higher-level runtimes build upon.
- *SimplePIM* [34]: A higher-level programming framework on UPMEM that provides array and iterator operators (e.g., `map`, `for_each`, `reduce`) together with lightweight communication primitives (e.g., `exchange`, `broadcast`, `gather`). SimplePIM abstracts much of the boilerplate of the raw SDK while preserving performance, effectively offering a *structured data-parallel layer* for PIM kernels.
- *PIM-STM* [195]: A software transactional memory (STM) runtime for UPMEM that provides transaction demarcation (begin/commit/abort), transactional read/write operations, and distributed conflict detection across DPUs. PIM-STM enables transaction-based concurrency control on PIM hardware, offering atomicity and consistency without requiring programmers to manually coordinate updates or reason about low-level synchronization.

Finally, treating **data locality as a first-class concern** remains a critical challenge. Because cross-module or cross-bank data fetches resurrect the "memory wall" in PIM settings, we argue for *affinity-aware* programming hooks. Although not originally designed for DRAM-PIM, two recent Near-Data Computing (NDC) efforts provide the *right interface ideas*: *Affinity Alloc* [272] captures data affinity in the allocation interface (e.g., a `pim_alloc(ptr, affinity_hint)` style), while *Leviathan* [238] offers a reactive, actor-based interface that unifies *what*, *where*, and *when* to compute. However, the **status quo** is that the public DRAM-PIM literature does *not* yet provide an affinity-aware, easy-to-use API that lets users declaratively bind data objects to PIM units and compose locality-aware execution.

5.2.2 Compiler and Offloading Decisions. **Goal:** Once the high-level workload is specified, the compiler partitions applications and generates optimized binaries for both host CPUs and DRAM-PIM units. Building on the previous section, where we discussed simulation frameworks and evaluation metrics for PIM systems, these compiler frameworks naturally leverage such metrics—e.g., bandwidth utilization, access locality, and compute intensity—to guide their deployment and optimization strategies. In other words, the performance indicators previously used for architectural evaluation now become inputs to *compiler-level decision making*.

Table 2. Representative Programming Interfaces for UPMEM

Category	Representative APIs	Purpose / Notes
UPMEM SDK (Host-side)	`dpu_alloc`, `dpu_free`, `dpu_load`, `dpu_copy_to`, `dpu_copy_from`, `dpu_launch`	Host orchestration and data movement.
UPMEM SDK (DPU-side)	`mem_alloc`, `mram_read`, `mram_write`, `barrier_wait`, `dpu_printf`	Local memory and MRAM access with basic sync.
Data-Parallel Abstractions	SimplePIM: Array/iterator primitives (`map`, `for_each`, `reduce`), communication operators (`exchange`, `broadcast`, `gather`)	Structured data-parallel and communication patterns.
Transactional Concurrency Runtime	PIM-STM: transactional begin/commit/abort, transactional read/write, validation mechanisms	Distributed STM-style concurrency control across DPUs.
Communication Collectives	PID-Comm: `reduce`, `all_reduce`, `broadcast`, `scatter`, `gather`, `barrier`	Distributed synchronization and collectives.

Automated Offloading Analysis. Modern compilers rely on analytical cost models that estimate the trade-off between computation and data movement. These models typically consider factors such as available parallelism, vectorization potential, working-set size, and interconnect contention. For instance, *To PIM or Not* introduces an LLVM-based framework that quantifies offloading benefit for bank-level RISC-V PIM and proposes lightweight vector extensions to improve kernel efficiency [62]. Building on this idea, A^3PIM integrates static analysis of memory-access patterns and cross-segment data transfers to enable analytic, automatic task placement across CPU and PIM back-ends, achieving average speedups of 2.63× over CPU-only and 4.45× over PIM-only baselines. [122].

Together, these studies illustrate a trend toward *compiler-driven, data-aware offloading*, where simulation-informed cost models replace manual kernel partitioning.

Hardware-Specific Code Generation. As DRAM-PIM architectures continue to diversify—from near-bank RISC-V cores to bit-serial SIMD and LUT-based designs—future compilers are expected to adopt an *adaptive, retargetable structure*: maintaining a unified high-level IR while supporting extensible, target-specific back-ends. Although most current toolchains remain tied to a single hardware design, early work such as CHOPPER demonstrates a first step toward this direction within the domain of bit-serial SIMD DRAM-PIM. It targets programmable bit-serial SIMD PuM architectures via specialized lowering passes that perform layout reordering, bit serialization, and instruction scheduling [227]. This evolving trend suggests that adaptive, IR-centric compilers will play a key role in bridging diverse DRAM-PIM ISAs and unifying their software ecosystems.

5.3 Runtime System Optimization

In DRAM-based Processing-in-Memory (PIM) architectures, **runtime system optimization** serves as the key bridge between compile-time design and real-time execution. Its necessity stems from three intrinsic properties of PIM systems. First, PIM introduces fundamentally new computation and communication paradigms—such as near-memory execution and customized ISA extensions—where online scheduling becomes an inherent part of the execution flow. Second, the mapping of data and tasks onto heterogeneous memory and compute resources forms a vast design space, where exhaustive offline mapping is often infeasible or suboptimal. Third, many workloads are inherently dynamic, as shown in PAPI[101], where workload characteristics vary across time, requiring adaptive runtime control.

Objective. The runtime system aims to dynamically manage heterogeneous resources, optimize performance, and ensure coherence during execution, particularly for cases that cannot be fully resolved at compile time. It operates as the adaptive layer coordinating CPUs, PIM units, and memory subsystems under changing workloads and system states.

Comparison to Offline Mapping. Unlike offline mapping, which assumes static workloads and pre-determined access patterns, runtime optimization continuously adapts to workload dynamics and system variations. Offline mapping is efficient for predictable, uniform workloads but fails under fluctuating load or dynamic input behavior. Runtime optimization instead monitors system states and dynamically adjusts task placement, data allocation, and coherence strategies, achieving better efficiency, adaptability, and robustness in practical DRAM-PIM environments.

Key Responsibilities. *(1) Supporting new execution paradigms.* As PIM architectures introduce novel ISAs and execution models, the runtime must bridge semantic gaps that static compilation alone cannot resolve. Emerging PIM frameworks, such as MetaNMP[33] and AsyncDIMM[38], redefine the division of labor between the host and near-memory compute units. These paradigms introduce new ISA semantics, synchronization models, and runtime decode mechanisms. The runtime system must manage instruction dispatch and synchronization among heterogeneous executors, enabling the asynchronous and data-centric execution patterns required by these designs.

(2) Dynamic task scheduling. The runtime scheduler in PIM systems must dynamically dispatch tasks across heterogeneous compute units—including the host CPU, PIM engines, and other cooperating hardware—while balancing **data affinity**, **load balance**, and resource constraints. Unlike traditional CPU/GPU schedulers, it operates under coupled compute–memory constraints and highly non-uniform data placement.

To handle these challenges, ABNMP[28] introduces an online cost model that jointly considers data locality and execution imbalance to guide task migration. Its runtime continuously monitors access latency and load distribution, performing lightweight rebalancing once the combined cost exceeds a threshold, achieving up to **1.47× speedup** and over 30% higher PIM utilization compared to static mapping.

Beyond intra-PIM balancing, the scheduler also decides the execution location of each task—on the CPU, PIM, or other accelerators—based on factors such as compute pattern, memory bandwidth sensitivity, and available capacity. PIMCloud[42] exemplifies this strategy by classifying tasks according to their compute/memory characteristics and dynamically mapping them to PIM or CPU resources while preserving QoS for latency-critical workloads. Overall, runtime scheduling functions as a real-time **decision engine** that adaptively balances locality, concurrency, and resource utilization across the heterogeneous PIM system.

(3) Data management. Because data placement directly determines performance, the runtime must adaptively reposition data as working sets shift or hot spots emerge. The runtime system handles placement and migration of data between standard DRAM and PIM-enabled memory regions. It determines when and where to replicate data to minimize access latency and congestion. As workloads evolve, dynamic data movement enables the system to maintain high locality without requiring full data remapping or synchronization at compile time.

(4) Memory and coherence management. As CPUs and PIM units access shared data, maintaining correctness with minimal communication becomes a core runtime responsibility, requiring the runtime to manage coherence across heterogeneous memory hierarchies. Systems such as CoNDA[25], LazyPIM[26], and PIM-MMU[160] introduce runtime-managed or hardware-assisted coherence mechanisms that ensure correctness with minimal overhead. These mechanisms rely on

the runtime layer to coordinate coherence events, translation tables, and synchronization between near-memory and host-side caches.

Overall, runtime system optimization acts as the intelligence core of DRAM-PIM architectures. By unifying scheduling, data management, and coherence control under a dynamic runtime layer, it enables truly adaptive, high-performance PIM systems that can balance locality, parallelism, and service quality under time-varying workloads.

5.4 Design Space Exploration (DSE) Tools

Compared with conventional accelerator design (e.g., GPUs or NPUs), Design Space Exploration (DSE) for DRAM-based Processing-in-Memory (PIM) presents unique challenges and opportunities. In typical hardware accelerators, computation and memory hierarchies are physically decoupled, and DSE mainly targets architectural balance between compute throughput, on-chip buffer sizing, and data reuse. In contrast, DRAM PIM integrates computation units directly into or near memory arrays, which tightly couples performance to memory access granularity, bank organization, and data placement. Consequently, PIM-oriented DSE must jointly reason about *memory device physics*, *dataflow mapping*, and *cross-bank communication*, making the exploration problem far more constrained but also richer in optimization potential.

Most existing PIM DSE frameworks share a common methodological foundation. They couple analytical or simulation-based performance, power, and area (PPA) models with heuristic or learning-driven search engines to traverse large multidimensional design spaces. These frameworks automatically assess trade-offs among parameters such as the number and type of compute units per bank, inter-bank interconnect topology, memory bandwidth partitioning, and scheduling strategies. Through automated search and evaluation, DSE tools reveal Pareto-optimal configurations that human designers may overlook, improving both design efficiency and hardware–software co-optimization fidelity. Conceptually, this exploration mindset is consistent with the compiler- and runtime-level techniques discussed earlier: offloading frameworks such as *To PIM or Not* and *A^3PIM* already employ analytic cost models to search over CPU–PIM partitioning, while runtime systems like ABNMP and PIMCloud use online models to adapt task placement and resource allocation on a fixed hardware substrate. In this subsection, however, we focus on *hardware-facing* co-design: the design space explicitly includes DRAM-PIM microarchitectural parameters (e.g., per-bank compute granularity, interconnect configuration, timing constraints) in addition to software mapping and scheduling, enabling end-to-end optimization that spans device physics up through workload dataflow.

Recent advances demonstrate a shift from architecture-centric tuning toward integrated, workload-aware exploration. For instance, *NMExplorer*[172] formulates a DSE framework for DIMM-based near-memory tensor reduction, balancing bandwidth utilization and reduction latency through exploring DIMM-level architectural configurations under different tensor-reduction scenarios. *SpecPIM*[174] extends this co-design philosophy to speculative large-language-model inference, combining architectural and dataflow search to optimize draft–verification pipelines under diverse model sizes. Similarly, *Bank on Compute-Near-Memory*[200] leverages a systematic PPA modeling framework to quantify area–energy–performance trade-offs of processing-near-bank architectures across DRAM standards, exposing scaling limits imposed by per-bank power density. In specialized workloads such as fully homomorphic encryption, *Affinity-based TFHE on PIM*[206] introduces affinity-aware mapping to minimize remote data accesses, achieving up to 209× speedup over CPU baselines by improving bank-locality in DRAM arrays.Among these, *NicePIM*[268] stands out as a representative 3D-stacked DRAM PIM framework: by combining a learning-assisted hardware tuner, a mapping enumerator, and an ILP-based scheduler, it attains an average **37% latency reduction**

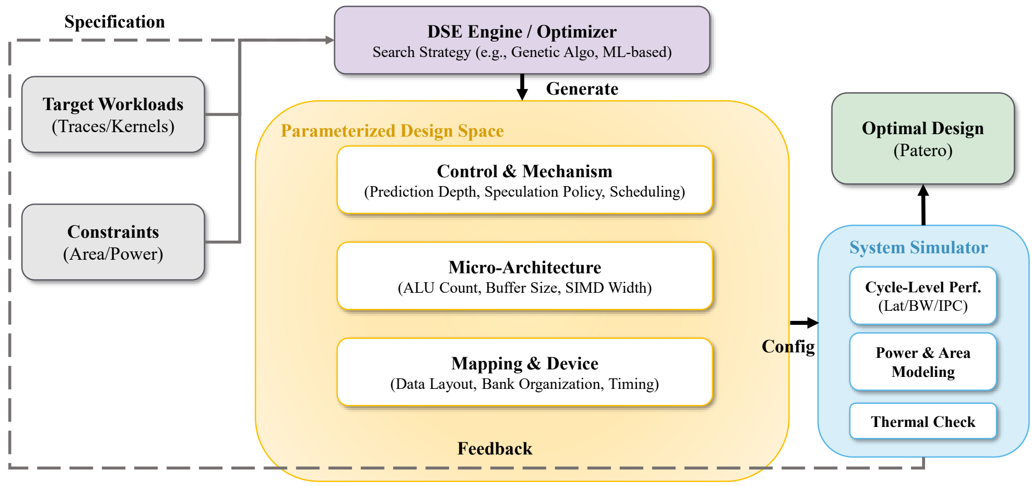

Fig. 12. Framework for PIM Design Space Exploration (DSE). Similar to methodologies used in SpecPIM and PIM-DSE, the process co-optimizes mechanism policies (e.g., speculation), architecture, and data mapping through cycle-accurate simulation.

and **28% energy savings** over fixed-architecture baselines, demonstrating the practical impact of holistic DSE.

Despite these advances, DRAM-PIM DSE still faces several open challenges. First, the coupling of memory timing, device variation, and logic placement makes accurate modeling difficult, often forcing researchers to rely on simplified assumptions. Second, the exponential growth of design parameters—from per-bank compute granularity to cross-layer dataflow mapping—poses severe scalability issues for search algorithms. Third, workload diversity further complicates exploration: LLMs, DNNs, embeddings, and encrypted computations exhibit distinct memory behaviors that demand workload-specific DSE formulations. Finally, integrating hardware-level DSE with system- and compiler-level optimization remains an open problem, as current frameworks rarely capture runtime scheduling, coherence, or host–PIM coordination effects. Addressing these challenges is essential for DSE to evolve from static design automation into a truly predictive and adaptive methodology for future DRAM-PIM systems.

6 Software-DRAM PIM co design and software optimization

6.1 Conceptual Foundations and Research Perspectives

The software and hardware co-design space for DRAM-based PIM is shaped by how the community conceptualizes the role of PIM within modern memory and compute hierarchies. Although the underlying hardware primitives are similar across platforms, researchers approach DRAM PIM from distinctly different vantage points, and each perspective implies a different software stack, programming model, and system integration strategy.

From an industry-centric perspective, most commercial efforts view DRAM PIM as an evolutionary extension of high-performance memory systems such as HBM and in-DRAM compute products including Samsung HBM-PIM or SK Hynix AiM. In this interpretation, PIM primarily functions as a bandwidth amplifier. Software support therefore centers on memory-centric optimizations such as bank-aware data placement, latency-conscious scheduling, and minimal extensions to existing memory controller or runtime interfaces. The emphasis is on achieving incremental acceleration without introducing disruptive changes to established programming abstractions.

Within the architectural research community, many works instead conceptualize DRAM PIM as a heterogeneous compute substrate that complements CPUs and GPUs. Under this model, PIM units are lightweight processing elements tightly coupled to DRAM banks. They are well suited for memory-bound operations while relying on the host processor for control-intensive or compute-heavy tasks. This framing motivates research on workload partitioning, scheduling, synchronization, and programming abstractions that integrate PIM into a broader heterogeneous multicore environment rather than treating it as a purely passive memory component.

Domain-focused and data-management researchers provide another viewpoint by treating PIM as an in-situ data processing layer. In database systems, PIM acts as a near-data execution engine for scans, filters, and reductions. In HTAP and cloud environments, PIM assists in managing latency-critical or bandwidth-heavy operations. For general-purpose acceleration, the software emphasis lies in portable models, compiler-level transformations, and adaptive runtimes. For domain-specific acceleration, emphasis shifts toward operator fusion, layout transformations, and application-driven dataflow scheduling.

At the algorithmic level, DRAM PIM can be modeled as a hybrid environment that blends shared-memory behavior at the CPU with distributed-memory characteristics across PIM banks. This encourages communication-avoiding formulations, locality-sensitive mappings, and the restructuring of computation so that it aligns with DRAM's internal parallelism and physical organization. The reference baseline chosen by a researcher, for example GPUs, CPUs, compute-in-memory arrays, or near-storage accelerators, also shapes how the tradeoffs of offloading and specialization are evaluated.

By establishing how different communities position DRAM-based PIM within the broader memory and compute hierarchy, this overview provides the foundation for understanding the diversity of software co-design strategies. The following sections build on this foundation by analyzing how representative application domains translate these conceptual viewpoints into specific dataflows, software abstractions, and hardware interactions.

6.2 Domain-Specific Software–Hardware Co-Design Practices

Building upon these diverse perspectives, we next examine how software–hardware co-design manifests across representative application domains, each exposing distinct dataflow patterns, memory access characteristics, and opportunities for DRAM-PIM acceleration. **For concrete examples illustrating these domain-specific practices, please refer to the Application Form in the Appendix.**

Deep Learning Workloads: From CNNs to LLMs

Deep learning workloads have evolved from the regular, compute-dense patterns of convolutional neural networks (CNNs) to the irregular and capacity-intensive behaviors of large language models (LLMs). Early CNNs feature dense linear algebra operations such as convolution and matrix–matrix multiplication, where DRAM-PIM acceleration mainly aims to exploit high internal bandwidth through weight-stationary or output-stationary dataflows. Software frameworks play a key role in operator fusion, tiling, and layout-aware scheduling, while the host CPU handles control flow and nonlinear activations.

In contrast, modern LLMs exhibit two distinct execution phases that expose heterogeneous performance characteristics. The *prefill* stage is compute-intensive, dominated by large matrix multiplications, whereas the *decode* stage is memory-bound due to key–value (KV) cache retrievals, embedding access, and attention operations. This natural separation motivates hybrid acceleration schemes that combine GPUs (for dense compute) with DRAM-PIM (for memory-intensive attention kernels). DRAM-PIM can offload softmax normalization, KV-cache prefetch, and matrix–vector multiplications, while GPUs sustain high-throughput tensor operations. Software runtimes orchestrate these heterogeneous units via compile-time graph partitioning and runtime profiling to dynamically tune offload ratios and overlap data movement with computation.

Beyond compute heterogeneity, LLMs also challenge the memory capacity limits of conventional GPU-based systems. The growing scale of model parameters and KV caches often exceeds on-package memory, motivating explorations into **CXL-PIM** and **memory disaggregation** architectures. In these designs, PIM-enabled DRAM modules are connected via Compute Express Link (CXL) to provide both capacity expansion and near-data processing capabilities. Software schedulers can then treat CXL-PIM as an extended memory tier capable of performing lightweight compute—such as cache maintenance, quantization, or partial reductions—close to data sources. This design alleviates memory pressure while preserving compute locality, forming a unified continuum from on-die GPU compute to near-memory and far-memory PIM execution.

Recommendation Systems: Modern recommendation models are characterized by large-scale embedding tables and sparse, data-dependent access patterns, which make them inherently memory-bound. The main bottleneck lies in embedding lookup and feature interaction layers, where frequent table accesses dominate both latency and bandwidth consumption. DRAM-PIM can serve as a near-memory accelerator by distributing embedding tables across banks to perform in-DRAM lookup, partial reduction, and lightweight arithmetic (e.g., addition, normalization) directly within memory arrays.

To complement DRAM-PIM, heterogeneous integration with content-addressable memory (CAM) or SRAM-based near-memory units can further accelerate small, associative operations such as index matching, feature gating, and hot-embedding caching. In such designs, CAM handles sparse key matching or frequently accessed embeddings, while DRAM-PIM executes large-table scanning and accumulation. The host processor orchestrates higher-level tasks such as aggregation, gradient updates, and global scheduling.

From a software perspective, effective acceleration requires fine-grained partitioning of embedding tables according to access frequency, adaptive mapping across DRAM and CAM domains, and skew-aware workload balancing to mitigate load imbalance. Runtime schedulers should minimize communication overhead between PIM and host, overlapping lookup, aggregation, and update phases. Such a co-design approach transforms the memory subsystem into an active participant in recommendation inference and training, rather than a passive data provider.

Privacy-Preserving Workloads (e.g., Fully Homomorphic Encryption, FHE): Fully homomorphic encryption (FHE) workloads involve massive numbers of bitwise logic and modular arithmetic operations that exhibit extremely low arithmetic intensity but high data movement. DRAM subarrays can naturally implement bit-level logic primitives such as AND, OR, and XOR through analog charge-sharing or digital row-activation mechanisms, making PIM a compelling substrate for accelerating ciphertext manipulation.

However, FHE ciphertexts are typically very large—often spanning tens or hundreds of kilobytes per ciphertext under practical parameter sets—posing serious challenges for in-DRAM mapping and data movement. To handle such large ciphertexts, recent DRAM-PIM designs adopt block-level partitioning and hierarchical mapping strategies: each ciphertext is split into fine-grained tiles distributed across multiple DRAM banks or subarrays. In-memory logic units operate on tiles independently and perform local modular accumulation before partial results are merged through inter-bank reduction. This structure exploits intrinsic DRAM parallelism while keeping data locality high.

From the software perspective, compiler-level partitioning and memory layout optimization are crucial to balance ciphertext size, subarray capacity, and access granularity. The runtime system further overlaps ciphertext transfers with ongoing PIM computation to hide host–memory latency. Together, these co-optimization techniques allow DRAM-PIM to process FHE workloads efficiently despite their large ciphertext footprint, achieving energy and bandwidth efficiency unattainable by traditional host-side execution.

Graph Analytics: Graph workloads, including traversal, PageRank, and GNN aggregation, are characterized by highly irregular memory access and low compute-to-memory ratios. DRAM-PIM architectures can exploit subarray-level parallelism to perform in-memory neighbor aggregation, frontier filtering, and partial sum accumulation without extensive data movement. Hybrid designs may combine near-memory and host processing: PIM performs fine-grained vertex or edge updates, while the host CPU manages global synchronization and control flow. Software orchestration must handle partitioning and scheduling based on graph sparsity and degree distribution, employing locality-aware mapping to minimize bank conflicts and idle cycles across PIM units.

Data-Intensive Analytics (e.g., Databases, Key–Value Stores, and Genomics): Data-intensive applications—including database query processing, key–value stores, and genomic analytics—exhibit large working sets, irregular access, and high data-movement costs, making them natural candidates for near-data acceleration. DRAM-PIM can execute scan, filter, and aggregation primitives directly within memory, reducing channel traffic and improving throughput, while associative modules such as CAM or near-bank logic complement DRAM-PIM by accelerating index matching, range queries, and key lookups. In relational and transactional databases, PIM can offload data-parallel operations such as selection, projection, and partial aggregation, while the host handles global joins, transaction coordination, and logging. Emerging hybrid OLTP/HTAP architectures exploit this split execution to reduce tail latency and contention: pointer-chasing, index traversal, and version-chain maintenance are performed near memory, whereas consistency control and commit phases remain host-resident. Beyond database systems, similar design philosophies apply to genomics and analytical pipelines. In genomics, sequence alignment, k-mer counting, and pattern matching can be recast into bitwise comparison or reduction operations suitable for in-DRAM execution. Across these domains, software co-design centers on query decomposition, data placement, and operator scheduling—deciding which predicates or stages to offload to PIM versus retain on the host. Such co-optimization transforms DRAM from a passive data repository into an active analytic substrate capable of bandwidth-efficient filtering, aggregation, and search.

Across diverse domains, the success of DRAM-PIM acceleration depends not only on the underlying hardware primitives but, more fundamentally, on how software restructures computation to align with memory organization and dataflow constraints. Effective co-design requires software to recognize the distinct performance regimes of each workload—such as compute- versus memory-bound phases in deep learning, skewed access patterns in recommendation systems, or bit-level arithmetic granularity in FHE—and to schedule, partition, and map data accordingly.

Instead of a one-size-fits-all runtime, domain-specific frameworks integrate compiler, memory layout, and runtime orchestration to exploit both locality and heterogeneity: partitioning computation across host and PIM, balancing data placement across banks or modules, and overlapping communication with in-memory execution. In this sense, software transforms DRAM from a passive storage medium into an active compute layer, closing the gap between algorithmic structure and physical data organization.

This co-design perspective also explains why modern architectures increasingly embrace **domain-specific accelerators**. By coupling workload-aware software stacks with memory-centric computing substrates such as DRAM-PIM or CXL-PIM, system designers can tailor the balance between compute, bandwidth, and capacity to the intrinsic characteristics of each application domain—achieving efficiency that general-purpose architectures cannot match.

7 Security in Processing-in-Memory

The Shift in Trust Model. **Why PIM security is distinct.** Many practical PIM designs (e.g., DIMM-resident UPMEM DPUs) reside *outside the CPU package and its TEE's TCB*. Offloading data and computation to such devices exposes *plaintext-in-use* beyond the CPU enclave boundary and enlarges the attack surface (physical access, DMA-like observation, and untrusted coprocessor risks) [81]. This architectural separation means that even when computation is encrypted at rest and in transit, PIM components can still access or manipulate intermediate plaintext states. In effect, the traditional CPU-centric trust model breaks down: integrity, confidentiality, and verifiability must be extended to the near-memory domain. Recent work explicitly treats PIM as an *untrusted in-memory accelerator* and designs secure offloading accordingly [81]. This redefinition of the threat boundary makes PIM security a first-class system concern rather than a peripheral hardware property.

We categorize the challenges and solutions into three logical layers: physical integrity, architectural isolation, and data confidentiality.

7.1 Physical Integrity: The RowHammer Challenge

Processing-using-DRAM (PuD) primitives exploit analog operations for high throughput, but they inherently stress the memory substrate.

Attacks (PuD-Induced Disturbance). PUD/PUM primitives (e.g., RowClone, SIMDRAM) exploit DRAM-internal analog operations to perform high-throughput in-array computation and bulk copy/initialize. Without proper throttling or counters, their highly localized activation and amplification patterns can approach RowHammer (RH) worst cases [97, 241]. Empirically, recent work demonstrates that **Processing-using-DRAM (PuD)** primitives themselves can induce *RowHammer-like read-disturbance* effects on real DRAM chips. **PuDHammer** systematically characterizes these phenomena, showing that certain in-DRAM bulk-activation patterns can trigger bit-flips under realistic activation rates [297]. This highlights that enabling PuD/PIM functionality without row-activation governance may reintroduce disturbance vulnerabilities even in modern devices. Conceptually, PuD-induced disturbance is not merely a reliability issue but a potential integrity and confidentiality breach: bit flips could corrupt cryptographic state or leak data via controlled fault injection.

Defenses (Array-side RH Governance). Beyond data-parallel computation, PUD can be *reused* to improve reliability and security. **P-PIM** performs *in-DRAM self-tracking and mitigation* of row activations to detect and reduce RH effects with low overhead [320]. At the industry level, DDR5 introduces *Per-Row Activation Counting (PRAC)* and *Alert Back-Off (ABO)* so DRAM can request targeted mitigation; controller-side designs like **BlockHammer** throttle aggressors via Bloom-filter tracking [196, 288]. Together, these approaches demonstrate a shift from post-factum detection to *proactive activation governance*, where both DRAM and controller collaborate to enforce safe activation budgets even under aggressive PIM traffic.

7.2 Architectural Isolation: Side Channels and System Boundaries

PIM introduces new active components on the shared memory hierarchy, blurring the isolation boundaries traditionally enforced by the OS and CPU.

Attacks (Bank/Rank Sharing and Side Channels). PIM introduces new high-bandwidth actors on the same DRAM hierarchy as the CPU. The IMPACT study shows high-throughput *covert/side channels* that exploit PIM-aware timing/resource contention and address-mapping properties [27]. These cross-domain channels can exist even when the CPU and PIM operate in disjoint address ranges because internal DRAM schedulers and refresh queues are shared. Consequently, PIM blurs the isolation boundaries that TEEs and OS mechanisms traditionally rely on, and naïve co-scheduling of host and PIM tasks may open new timing-based information flows.

System Boundary Risks. Memory-encryption domains, IOMMU page tables, and coherence/consistency mechanisms can be stressed by concurrent CPU–PIM fetch/update. If not mediated by a trusted controller or OS path, these interactions may enable integrity or rollback issues [27]. Such races can silently break assumptions held by memory-encryption engines or transactional systems. Therefore, a trustworthy PIM integration requires the host to enforce ordering, versioning, and domain-aware access control to prevent unintended cross-domain interference.

7.3 Data Confidentiality: Trusted Execution vs. Cryptography

Perhaps the most critical challenge is protecting *computation* on PIM devices that lack a verifiable root-of-trust or physical tamper resistance. Challenges here stem from device trust issues, while solutions divide into hardware-based and cryptography-based approaches.

Challenge: Device/Firmware Trust. Real deployments such as the **UPMEM** DIMM-based PIM system introduce a new *firmware and attestation surface*. Each DPU maintains its own boot image, runtime, and update path that reside outside the CPU's trusted computing base, yet interact with host memory through DMA-like channels. Public documentation of current UPMEM hardware does not disclose a complete secure-boot or remote-attestation mechanism. This highlights that DIMM-resident DPUs expand the system's attack surface orthogonally to CPU TEEs, requiring device-level measurement and isolation primitives. In a broader context, this mirrors historical challenges seen in *GPU and NIC offloading:* lack of verifiable firmware provenance can undermine system-wide guarantees even if higher layers remain formally verified.

Solution A: Trustworthy PIM Hardware (Enclaves). Designs like **PIM-Enclave** and **SE-PIM** advocate for extending the TCB into the memory module by bringing attestation, isolation, and side-channel-aware execution into the memory/logic layer [66, 67]. These designs argue that trust should be anchored near the data. In practice, their success depends on whether emerging 3D-stacked memory vendors are willing to expose logic-layer root-of-trust primitives.

Solution B: Cryptographic Protection (MPC & FHE). An alternative is to treat the PIM device as strictly *untrusted.* **SecUPMEM** co-designs arithmetic secret sharing and Yao garbling to split CPU/PIM work, so the PIM observes only masked shares while still delivering bandwidth benefits [81]. This illustrates that cryptography and architecture can co-exist: computation bandwidth is preserved while privacy is mathematically guaranteed. *However, a key research challenge is generalizing such secure offloading to heterogeneous PIM fabrics (e.g., multiple DIMMs) without prohibitive synchronization overhead.* Furthermore, when viable, **Software-level Homomorphic Compute (FHE)** allows computation directly on encrypted data, though typically limited to narrower kernels. Integrating FHE or MPC with PIM raises a new trade-off frontier: whether to prioritize absolute trust minimization or throughput, depending on workload sensitivity.

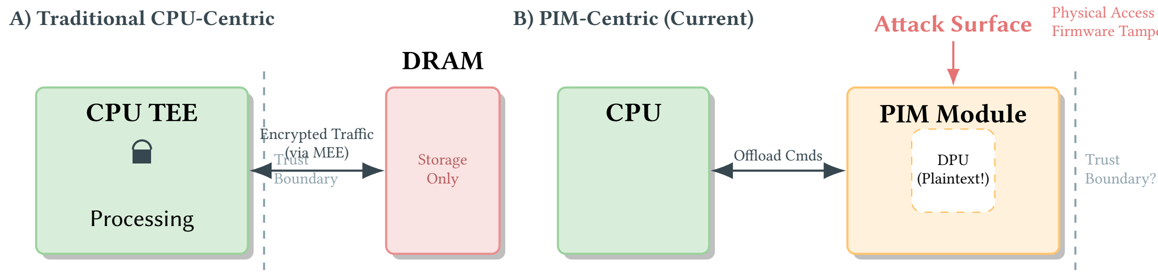

Fig. 13. Comparison of Trust Models. (A) In TEE-enabled CPUs (e.g., SGX), data on the bus is encrypted by hardware engines (MEE). (B) In PIM, computation occurs on the module, exposing plaintext to physical and firmware attacks.

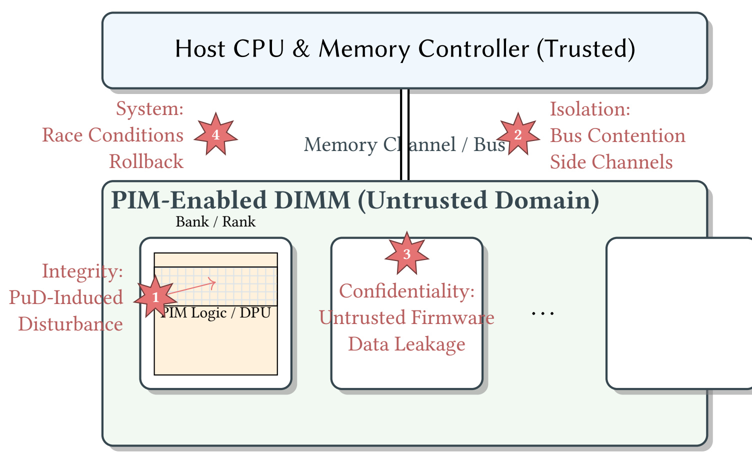

Fig. 14. Anatomy of Security Vulnerabilities in PIM. (1) Physical substrate disturbance; (2) Shared channel leakage; (3) Device trust issues; (4) System-level consistency races.

Takeaways. (1) Treat PIM/PUD as a *new trust domain* with explicit plaintext-in-use exposure; (2) couple PUD enablement with array-side RH governance (PRAC/P-PIM/BlockHammer) and bank/rank traffic shaping; (3) pick either *cryptographic protection* (MPC/FHE) for untrusted devices or *enclave-like* designs for trusted PIM. Beyond these immediate lessons, a broader implication is that secure PIM design will likely mirror the evolution of CPU TEEs: starting from coarse-grained isolation and moving toward composable, verifiable, and attested execution close to memory. Achieving this demands joint advances in DRAM architecture, firmware governance, and secure computation protocols.

8 Future Directions and Open Challenges

While DRAM-based PIM has matured from theoretical proposals to real-world silicon prototypes (e.g., UPMEM, Samsung HBM-PIM), widespread adoption still faces significant hurdles. The next phase of PIM research must shift focus from pure architectural innovation to *system-level integration*, *standardization*, and *robustness*. We highlight four critical directions for future investigation.

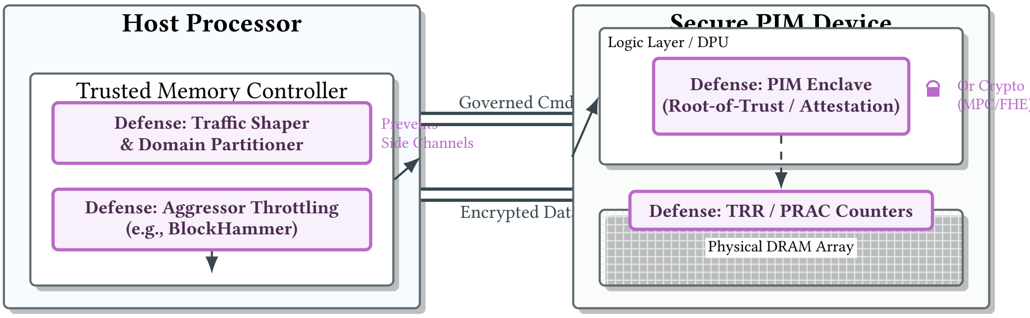

Fig. 15. Proposed Secure-PIM Architecture. Defenses are embedded at specific hardware points: (Left) Host controller enforces traffic shaping and throttling; (Right) PIM device integrates Enclaves for confidentiality and array-side counters for physical integrity.

8.1 Standardization and CXL-Enabled Disaggregation

Currently, most PIM solutions rely on proprietary ISAs and custom physical interfaces, creating a "vendor lock-in" dilemma that hinders ecosystem growth.

CXL-PIM Convergence. The emergence of Compute Express Link (CXL) offers a unified interface for memory disaggregation. A critical research frontier is integrating PIM capabilities into CXL Type-2 or Type-3 devices (CXL-PIM). Future work must explore how PIM operates over CXL.mem protocols, handling latency, coherency, and the potential for "Near-Data Processing" within CXL memory expanders without modifying the host memory controller.

Unified ISA and API Standards. To avoid fragmentation, the community (e.g., JEDEC, RISC-V) needs to define a standardized subset of PIM instructions (e.g., bulk bitwise operations, vector addition) and uniform APIs (like OpenCL or OpenMP extensions for PIM). This would allow software developers to write code once and run it across varying PIM hardware (e.g., switching between HBM-PIM and DIMM-PIM).

8.2 Holistic Software Stack: From Compiler to OS

The "software gap" remains the primary bottleneck for commercialization. Existing approaches often require manual code rewriting or intrinsic-based programming, which is impractical for large-scale deployment.

Transparent Compilation and Automatic Offloading. Future compilers need sophisticated cost models to automatically identify code regions suitable for PIM (e.g., memory-bound, low-locality kernels) and handle data layout transformation (e.g., row-interleaved to bank-partitioned) transparently.

OS-Level Resource Management. PIM introduces heterogeneous compute resources that the OS must manage. Open challenges include PIM-aware virtual memory management (handling huge pages and physical contiguity requirements), multi-tenant scheduling (preventing PIM resource contention), and seamless context switching between host and PIM execution.

8.3 PIM for Emerging GenAI Workloads

While PIM excels at traditional bandwidth-bound tasks, the explosion of Generative AI (LLMs) presents new opportunities and challenges.

KV-Cache Offloading. In Large Language Model (LLM) inference, the auto-regressive decoding phase is heavily memory-bandwidth bound due to the KV-cache. Designing PIM architectures specialized for KV-cache management and attention calculation (e.g., near-bank Softmax and quantization) is a high-value direction.

Dynamic Sparsity Support. Modern AI workloads exploit unstructured sparsity. Traditional PIM designs (SIMD/SIMT) struggle with irregular memory access patterns. Future "Smart PIM" designs should incorporate lightweight gather-scatter units or indirect access mechanisms to efficiently support sparse matrix operations (SpMV) and Graph Neural Networks (GNNs).

8.4 Reliability and Thermal Feasibility

Moving computation into DRAM introduces thermal and reliability issues that are often overlooked in architectural simulations.

Thermal-Aware Scheduling. Logic operations generate significant heat, potentially degrading DRAM data retention times. Future systems require hardware-software co-design for thermal throttling, where the OS or controller dynamically migrates "hot" pages or throttles PIM frequency to maintain signal integrity.

Fault Tolerance in 3D Stacks. For 3D-stacked PIM (e.g., HBM-based), the yield and reliability of Through-Silicon Vias (TSVs) are critical. Research is needed on low-overhead ECC schemes that can protect both data storage and intermediate computation results without imposing excessive area penalties on the logic layer.

9 Appendix: Application Summary

Table 3. Summary of DRAM-PIM Works by Application Domain and Task

Application Domain	Specific Task	Representative Works
General-Purpose	General-purpose workloads	[4, 6, 10, 13, 18, 25, 29, 32, 34, 37, 38, 48, 50, 55, 61, 62, 68, 69, 71, 72, 75–79, 84, 89, 95–98, 107–109, 114, 122, 123, 125, 125, 128, 129, 131, 137, 139, 143, 148, 151, 157, 160, 164, 165, 167, 170, 175, 176, 180, 186, 192, 195, 200, 202, 204, 207, 208, 211–214, 227, 229, 234, 235, 238, 242, 245, 246, 248, 250, 252, 255, 259, 261, 262, 264, 265, 271–273, 277, 278, 281, 283–287, 292–294, 296, 297, 303, 304, 306, 310, 314, 315, 317, 319, 320, 322, 326]
AI / Machine Learning	Large Language Models (LLMs)	Heterogeneous[15, 41, 64, 101, 102, 119, 146, 150, 162, 174, 183, 194, 219, 239, 240, 300, 318], [313],CXL[15, 93, 222],compiler[120, 280], pruning [217], undecide[2, 15, 155, 215, 218],RAG[187],hybrrid-bonding[295], circuit[163]

Continued on next page

Table 3. Summary of DRAM-PIM Works (Continued)

Application Domain	Specific Task	Representative Works
	Traditional Neural Networks (DNN, GEMM, etc.)	[3, 31, 33, 35, 36, 46, 47, 49, 51, 52, 56–60, 70, 79, 80, 82, 83, 87, 90, 92, 100, 103, 105, 106, 115, 121, 124, 136, 138, 140, 141, 145, 149, 153, 154, 156, 158, 166, 169, 173, 178, 185, 199, 201, 254, 256, 257, 268, 269, 274, 279, 282, 299, 302, 309, 323]
Graph Processing	Graph Analytics (BFS, SSSP, etc.) and Graph Neural Networks (GNNs)	[1, 5, 8, 14, 23, 30, 41, 53, 85, 99, 181, 182, 205, 258, 270, 275, 307, 316, 321, 325]
Data-Intensive Analytics	Recommendation Systems	CXL[15, 177, 190],heterogeneous[132, 289, 324],training[188, 198],real implementation[15, 40, 43, 44, 117, 133, 152, 168, 276],parallelism[189],undecided[290, 291],3D-IC[209],tensor reduction[220]
	Databases and OLAP/OLTP Workloads	[15, 22, 134, 142, 184]
Security & Reliability	Cryptography (e.g., FHE and related tasks)	[45, 86, 94, 126, 130, 135, 144, 178, 191, 206, 223, 224, 237, 249]
	DRAM Security and Reliability (e.g., RowHammer)	[16, 27, 66, 67, 81, 196, 297, 320]
Other Specialized Computing	Bioinformatics, Genome Analysis, Networking, etc.	[7, 9, 11, 12, 17, 19–22, 39, 42, 63, 65, 65, 73, 74, 91, 104, 110–113, 118, 127, 134, 147, 159, 172, 184, 193, 197, 203, 221, 225, 226, 228, 230, 231, 236, 243, 244, 251, 260, 263, 266, 305, 308, 311, 312]

References

[1] Abraham Addisie and Valeria Bertacco. 2020. Centaur: Hybrid processing in on/off-chip memory architecture for graph analytics. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.

[2] Salma Afifi, Ishan Thakkar, and Sudeep Pasricha. 2024. ARTEMIS: A Mixed Analog-Stochastic In-DRAM Accelerator for Transformer Neural Networks. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 43, 11 (2024), 3336–3347.

[3] Shaizeen Aga, Nuwan Jayasena, and Mike Ignatowski. 2019. Co-ML: a case for Co llaborative ML acceleration using near-data processing. In *Proceedings of the International Symposium on Memory Systems*. 506–517.

[4] Hameeza Ahmed, Paulo C Santos, João PC Lima, Rafael F Moura, Marco AZ Alves, Antônio CS Beck, and Luigi Carro. 2019. A compiler for automatic selection of suitable processing-in-memory instructions. In *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 564–569.

[5] Junwhan Ahn, Sungpack Hong, Sungjoo Yoo, Onur Mutlu, and Kiyoung Choi. 2015. A scalable processing-in-memory accelerator for parallel graph processing. In *Proceedings of the 42nd annual international symposium on computer architecture*. 105–117.

[6] Junwhan Ahn, Sungjoo Yoo, Onur Mutlu, and Kiyoung Choi. 2015. PIM-enabled instructions: A low-overhead, locality-aware processing-in-memory architecture. *ACM SIGARCH Computer Architecture News* 43, 3S (2015), 336–348.

[7] Berkin Akin, Franz Franchetti, and James C Hoe. 2015. Data reorganization in memory using 3D-stacked DRAM. *ACM SIGARCH Computer Architecture News* 43, 3S (2015), 131–143.

[8] Mustafa F Ali, Akhilesh Jaiswal, and Kaushik Roy. 2019. In-memory low-cost bit-serial addition using commodity DRAM technology. *IEEE Transactions on Circuits and Systems I: Regular Papers* 67, 1 (2019), 155–165.

[9] Mohammad Alian and Nam Sung Kim. 2019. Netdimm: Low-latency near-memory network interface architecture. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*. 699–711.

[10] Mohammad Alian, Seung Won Min, Hadi Asgharimoghaddam, Ashutosh Dhar, Dong Kai Wang, Thomas Roewer, Adam McPadden, Oliver O'Halloran, Deming Chen, Jinjun Xiong, et al. 2018. Application-transparent near-memory processing architecture with memory channel network. In *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 802–814.

[11] Yuda An, Yunxiao Tang, Shushu Yi, Li Peng, Xiurui Pan, Guangyu Sun, Zhaochu Luo, Qiao Li, and Jie Zhang. 2024. StreamPIM: Streaming matrix computation in racetrack memory. In *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 297–311.

[12] Shaahin Angizi, Naima Ahmed Fahmi, Wei Zhang, and Deliang Fan. 2020. Pim-assembler: A processing-in-memory platform for genome assembly. In *2020 57th ACM/IEEE design automation conference (DAC)*. IEEE, 1–6.

[13] Shaahin Angizi and Deliang Fan. 2019. Redram: A reconfigurable processing-in-dram platform for accelerating bulk bit-wise operations. In *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 1–8.

[14] Md Tanvir Arafin. 2022. Computation-in-Memory Accelerators for Secure Graph Database: Opportunities and Challenges. In *2022 27th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 31–36.

[15] Md Tanvir Arafin and Zhaojun Lu. 2020. Security challenges of processing-in-memory systems. In *Proceedings of the 2020 on Great Lakes Symposium on VLSI*. 229–234.

[16] Md Tanvir Arafin and Zhaojun Lu. 2020. Security challenges of processing-in-memory systems. In *Proceedings of the 2020 on Great Lakes Symposium on VLSI*. 229–234.

[17] Bahar Asgari, Ramyad Hadidi, Jiashen Cao, Da Eun Shim, Sung-Kyu Lim, and Hyesoon Kim. 2021. Fafnir: Accelerating sparse gathering by using efficient near-memory intelligent reduction. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 908–920.

[18] Hadi Asghari-Moghaddam, Young Hoon Son, Jung Ho Ahn, and Nam Sung Kim. 2016. Chameleon: Versatile and practical near-DRAM acceleration architecture for large memory systems. In *2016 49th annual IEEE/ACM international symposium on Microarchitecture (MICRO)*. IEEE, 1–13.

[19] Daehyeon Baek, Soojin Hwang, and Jaehyuk Huh. 2024. pSyncPIM: Partially Synchronous Execution of Sparse Matrix Operations for All-Bank PIM Architectures. In *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 354–367.

[20] Alexander Baumstark, Muhammad Attahir Jibril, and Kai-Uwe Sattler. 2023. Accelerating large table scan using processing-in-memory technology. *Datenbank-Spektrum* 23, 3 (2023), 199–209.

[21] Alexander Baumstark, Muhammad Attahir Jibril, and Kai-Uwe Sattler. 2023. Processing-in-memory for databases: Query processing and data transfer. In *Proceedings of the 19th International Workshop on Data Management on New Hardware*. 107–111.

[22] Arthur Bernhardt, Andreas Koch, and Ilia Petrov. 2023. Pimdb: From main-memory dbms to processing-in-memory dbms-engines on intelligent memories. In *Proceedings of the 19th International Workshop on Data Management on New Hardware*. 44–52.

[23] Maciej Besta, Raghavendra Kanakagiri, Grzegorz Kwasniewski, Rachata Ausavarungnirun, Jakub Beránek, Konstantinos Kanellopoulos, Kacper Janda, Zur Vonarburg-Shmaria, Lukas Gianinazzi, Ioana Stefan, et al. 2021. Sisa: Set-centric instruction set architecture for graph mining on processing-in-memory systems. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*. 282–297.

[24] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R Hower, Tushar Krishna, Somayeh Sardashti, et al. 2011. The gem5 simulator. *ACM SIGARCH computer architecture news* 39, 2 (2011), 1–7.

[25] Amirali Boroumand, Saugata Ghose, Minesh Patel, Hasan Hassan, Brandon Lucia, Rachata Ausavarungnirun, Kevin Hsieh, Nastaran Hajinazar, Krishna T Malladi, Hongzhong Zheng, et al. 2019. CoNDA: Efficient cache coherence support for near-data accelerators. In *Proceedings of the 46th International Symposium on Computer Architecture*. 629–642.

[26] Amirali Boroumand, Saugata Ghose, Minesh Patel, Hasan Hassan, Brandon Lucia, Kevin Hsieh, Krishna T Malladi, Hongzhong Zheng, and Onur Mutlu. 2016. LazyPIM: An efficient cache coherence mechanism for processing-in-memory. *IEEE Computer Architecture Letters* 16, 1 (2016), 46–50.

[27] F Nisa Bostancı, Konstantinos Kanellopoulos, Ataberk Olgun, A Giray Yağlıkçı, İsmail Emir Yüksel, Nika Mansouri Ghiasi, Zülal Bingöl, Mohammad Sadrosadati, and Onur Mutlu. 2025. Revisiting Main Memory-Based Covert and Side Channel Attacks in the Context of Processing-in-Memory. In *2025 55th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 16–32.

[28] Johannes L. Braams and Javier Bezos. 2022. *Babel*. http://www.ctan.org/pkg/babel.

[29] Jingwei Cai, Yuchen Wei, Zuotong Wu, Sen Peng, and Kaisheng Ma. 2023. Inter-layer scheduling space definition and exploration for tiled accelerators. In *Proceedings of the 50th Annual International Symposium on Computer Architecture*. 1–17.

[30] Shuangyu Cai, Boyu Tian, Huanchen Zhang, and Mingyu Gao. 2024. Pimpam: Efficient graph pattern matching on real processing-in-memory hardware. *Proceedings of the ACM on Management of Data* 2, 3 (2024), 1–25.

[31] Muya Chang, Ashwin Sanjay Lele, Samuel D Spetalnick, Brian Crafton, Shota Konno, Zishen Wan, Ashwin Bhat, Win-San Khwa, Yu-Der Chih, Meng-Fan Chang, et al. 2023. A 73.53 TOPS/W 14.74 TOPS heterogeneous RRAM in-memory and SRAM near-memory SoC for hybrid frame and event-based target tracking. In *2023 IEEE International Solid-State Circuits Conference (ISSCC)*. IEEE, 426–428.

[32] Tung-Cheng Chang, Yen-Cheng Chiu, Chun-Ying Lee, Je-Min Hung, Kuang-Tang Chang, Cheng-Xin Xue, Ssu-Yen Wu, Hui-Yao Kao, Peng Chen, Hsiao-Yu Huang, et al. 2020. 13.4 A 22nm 1Mb 1024b-read and near-memory-computing dual-mode STT-MRAM macro with 42.6 GB/s read bandwidth for security-aware mobile devices. In *2020 IEEE International Solid-State Circuits Conference-(ISSCC)*. IEEE, 224–226.

[33] Dan Chen, Haiheng He, Hai Jin, Long Zheng, Yu Huang, Xinyang Shen, and Xiaofei Liao. 2023. Metanmp: Leveraging cartesian-like product to accelerate hgnns with near-memory processing. In *Proceedings of the 50th Annual International Symposium on Computer Architecture*. 1–13.

[34] Jinfan Chen, Juan Gómez-Luna, Izzat El Hajj, Yuxin Guo, and Onur Mutlu. 2023. Simplepim: A software framework for productive and efficient processing-in-memory. In *2023 32nd International Conference on Parallel Architectures and Compilation Techniques (PACT)*. IEEE, 99–111.

[35] Jiaxian Chen, Yiquan Lin, Kaoyi Sun, Jiexin Chen, Chenlin Ma, Rui Mao, and Yi Wang. 2022. GCIM: Towards Efficient Processing of Graph Convolutional Networks in 3D-Stacked Memory. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2022).

[36] Jiaxian Chen, Zhaoyu Zhong, Kaoyi Sun, Chenlin Ma, Rui Mao, and Yi Wang. 2023. Lift: Exploiting hybrid stacked memory for energy-efficient processing of graph convolutional networks. In *2023 60th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.

[37] Liyan Chen, Jianfei Jiang, Qin Wang, Zhigang Mao, and Naifeng Jing. 2024. Bridge-ndp: Achieving efficient communication-computation overlap in near data processing with bridge architecture. In *2024 29th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 460–465.

[38] Liyan Chen, Dongxu Lyu, Jianfei Jiang, Qin Wang, Zhigang Mao, and Naifeng Jing. 2025. AsyncDIMM: Achieving Asynchronous Execution in DIMM-Based Near-Memory Processing. In *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 518–532.

[39] Liang-Chi Chen, Chien-Chung Ho, and Yuan-Hao Chang. 2023. Uppipe: A novel pipeline management on in-memory processors for rna-seq quantification. In *2023 60th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.

[40] Mingkai Chen, Tianhua Han, Cheng Liu, Shengwen Liang, Kuai Yu, Lei Dai, Ziming Yuan, Ying Wang, Lei Zhang, Huawei Li, et al. 2024. DRIM-ANN: An Approximate Nearest Neighbor Search Engine based on Commercial DRAM-PIMs. *arXiv preprint arXiv:2410.15621* (2024).

[41] Ruiyang Chen, Zhuoran Song, Yicheng Zheng, Zeyu Zhu, Gang Li, Naifeng Jing, Xiaoyao Liang, and Haibing Guan. 2025. HEAT: NPU-NDP HE terogeneous A rchitecture for T ransformer-Empowered Graph Neural Networks. In *Proceedings of the 58th IEEE/ACM International Symposium on Microarchitecture®*. 263–276.

[42] Shuang Chen, Yi Jiang, Christina Delimitrou, and José F Martínez. 2022. PIMCloud: QoS-aware resource management of latency-critical applications in clouds with processing-in-memory. In *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 1086–1099.

[43] Sitian Chen, Haobin Tan, Amelie Chi Zhou, Yusen Li, and Pavan Balaji. 2024. Updlrm: Accelerating personalized recommendation using real-world pim architecture. In *Proceedings of the 61st ACM/IEEE Design Automation Conference*. 1–6.

[44] Sitian Chen, Amelie Chi Zhou, Yucheng Shi, Yusen Li, and Xin Yao. 2024. MemANNS: Enhancing Billion-Scale ANNS Efficiency with Practical PIM Hardware. *arXiv preprint arXiv:2410.23805* (2024).

[45] Zehao Chen, Zhining Cao, Zhaoyan Shen, and Lei Ju. 2024. HMC-FHE: A Heterogeneous Near Data Processing Framework for Homomorphic Encryption. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 43, 11 (2024), 3551–3563.

[46] Yen-Cheng Chiu, Win-San Khwa, Chung-Yuan Li, Fang-Ling Hsieh, Yu-An Chien, Guan-Yi Lin, Po-Jung Chen, Tsen-Hsiang Pan, De-Qi You, Fang-Yi Chen, et al. 2023. A 22nm 8Mb STT-MRAM near-memory-computing macro with

8b-precision and 46.4-160.1 TOPS/W for edge-AI devices. In *2023 IEEE International Solid-State Circuits Conference (ISSCC)*. IEEE, 496–498.

[47] Benjamin Y Cho, Jeageun Jung, and Mattan Erez. 2021. Accelerating bandwidth-bound deep learning inference with main-memory accelerators. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–14.

[48] Benjamin Y Cho, Yongkee Kwon, Sangkug Lym, and Mattan Erez. 2020. Near data acceleration with concurrent host access. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 818–831.

[49] Seunghwan Cho, Haerang Choi, Eunhyeok Park, Hyunsung Shin, and Sungjoo Yoo. 2020. McDRAM v2: In-dynamic random access memory systolic array accelerator to address the large model problem in deep neural networks on the edge. *IEEE Access* 8 (2020), 135223–135243.

[50] Duheon Choi, Taeyang Jeong, Joonhyeok Yeom, and Eui-Young Chung. 2023. Operand-Oriented Virtual Memory Support for Near-Memory Processing. *IEEE Trans. Comput.* 72, 8 (2023), 2250–2263.

[51] Jungwoo Choi, Hyuk-Jae Lee, and Chae Eun Rhee. 2022. ADC-PIM: Accelerating convolution on the GPU via in-memory approximate data comparison. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 12, 2 (2022), 458–471.

[52] 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)*. IEEE, 1–9.

[53] Guohao Dai, Tianhao Huang, Yuze Chi, Jishen Zhao, Guangyu Sun, Yongpan Liu, Yu Wang, Yuan Xie, and Huazhong Yang. 2018. GraphH: A processing-in-memory architecture for large-scale graph processing. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 38, 4 (2018), 640–653.

[54] Guohao Dai, Zhenhua Zhu, Tianyu Fu, Chiyue Wei, Bangyan Wang, Xiangyu Li, Yuan Xie, Huazhong Yang, and Yu Wang. 2022. Dimmining: pruning-efficient and parallel graph mining on near-memory-computing. In *Proceedings of the 49th Annual International Symposium on Computer Architecture*. 130–145.

[55] Zhuoyu Dai, Feibin Xiang, Xiangqu Fu, Yifan He, Wenyu Sun, Yongpan Liu, Guanhua Yang, Feng Zhang, Jinshan Yue, and Ling Li. 2024. A Multichiplet Computing-in-Memory Architecture Exploration Framework Based on Various CIM Devices. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 43, 12 (2024), 4613–4625.

[56] Palash Das, Ajay Joshi, and Hemangee K Kapoor. 2022. Hydra: A near hybrid memory accelerator for cnn inference. In *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 1017–1022.

[57] Palash Das and Hemangee K Kapoor. 2020. Nzespa: A near-3d-memory zero skipping parallel accelerator for cnns. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 40, 8 (2020), 1573–1585.

[58] Palash Das, Shashank Sharma, and Hemangee K Kapoor. 2022. ALAMNI: Adaptive LookAside memory based near-memory inference engine for eliminating multiplications in real-time. *IEEE Trans. Comput.* 72, 3 (2022), 693–706.

[59] Prangon Das, Purab Ranjan Sutradhar, Mark Indovina, Sai Manoj Pudukotai Dinakarrao, and Amlan Ganguly. 2022. Implementation and evaluation of deep neural networks in commercially available processing in memory hardware. In *2022 IEEE 35th International System-on-Chip Conference (SOCC)*. IEEE, 1–6.

[60] 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 *Proceedings of the 56th Annual Design Automation Conference 2019*. 1–6.

[61] Fabrice Devaux. 2019. The true processing in memory accelerator. In *2019 IEEE Hot Chips 31 Symposium (HCS)*. IEEE Computer Society, 1–24.

[62] Alexandar Devic, Siddhartha Balakrishna Rai, Anand Sivasubramaniam, Ameen Akel, Sean Eilert, and Justin Eno. 2022. To pim or not for emerging general purpose processing in ddr memory systems. In *Proceedings of the 49th Annual International Symposium on Computer Architecture*. 231–244.

[63] Safaa Diab, Amir Nassereldine, Mohammed Alser, Juan Gómez Luna, Onur Mutlu, and Izzat El Hajj. 2023. A framework for high-throughput sequence alignment using real processing-in-memory systems. *Bioinformatics* 39, 5 (2023), btad155.

[64] Yan Ding, Chubo Liu, Mingxing Duan, Wanli Chang, Keqin Li, and Kenli Li. 2023. HAIMA: A hybrid SRAM and DRAM accelerator-in-memory architecture for transformer. In *2023 60th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.

[65] Ayushi Dube, Ankit Wagle, Gian Singh, and Sarma Vrudhula. 2022. Tunable precision control for approximate image filtering in an in-memory architecture with embedded neurons. In *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*. 1–9.

[66] Kha Dinh Duy and Hojoon Lee. 2021. PIM-Enclave: Bringing Confidential Computation Inside Memory. *arXiv preprint arXiv:2111.03307* (2021).

[67] Kha Dinh Duy and Hojoon Lee. 2022. SE-PIM: in-memory acceleration of data-intensive confidential computing. *IEEE Transactions on Cloud Computing* 11, 3 (2022), 2473–2490.

[68] Yasuko Eckert, Nuwan Jayasena, and Gabriel H Loh. 2014. Thermal feasibility of die-stacked processing in memory. In *2nd Workshop on Near-Data Processing (WoNDP)*.

[69] Pouya Esmaili-Dokht, Miquel Guiot, Petar Radojkovicć, Xavier Martorell, Eduard Ayguadé, Jesus Labarta, Jason Adlard, Paolo Amato, and Marco Sforzin. 2024. O(n) Key-Value Sort With Active Compute Memory. *IEEE Trans. Comput.* 73, 5 (2024), 1341–1356.

[70] Renhao Fan, Yikai Cui, Qilin Chen, Mingyu Wang, Youhui Zhang, Weimin Zheng, and Zhaolin Li. 2023. Maicc: A lightweight many-core architecture with in-cache computing for multi-dnn parallel inference. In *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*. 411–423.

[71] Amin Farmahini-Farahani, Jung Ho Ahn, Katherine Morrow, and Nam Sung Kim. 2015. NDA: Near-DRAM acceleration architecture leveraging commodity DRAM devices and standard memory modules. In *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 283–295.

[72] Amel Fatima, Sihang Liu, Korakit Seemakhupt, Rachata Ausavarungnirun, and Samira Khan. 2023. vPIM: Efficient virtual address translation for scalable processing-in-memory architectures. In *2023 60th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.

[73] Siying Feng, Xin He, Kuan-Yu Chen, Liu Ke, Xuan Zhang, David Blaauw, Trevor Mudge, and Ronald Dreslinski. 2022. MeNDA: A near-memory multi-way merge solution for sparse transposition and dataflows. In *Proceedings of the 49th Annual International Symposium on Computer Architecture*. 245–258.

[74] Ivan Fernandez, Ricardo Quislant, Eladio Gutiérrez, Oscar Plata, Christina Giannoula, Mohammed Alser, Juan Gómez-Luna, and Onur Mutlu. 2020. NATSA: A near-data processing accelerator for time series analysis. In *2020 IEEE 38th International Conference on Computer Design (ICCD)*. IEEE, 120–129.

[75] João Dinis Ferreira, Gabriel Falcao, Juan Gómez-Luna, Mohammed Alser, Lois Orosa, Mohammad Sadrosadati, Jeremie S Kim, Geraldo F Oliveira, Taha Shahroodi, Anant Nori, et al. 2022. pLUTo: Enabling massively parallel computation in DRAM via lookup tables. In *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 900–919.

[76] Daichi Fujiki. 2023. Mvc: Enabling fully coherent multi-data-views through the memory hierarchy with processing in memory. In *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*. 800–814.

[77] 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*. 100–113.

[78] Mingyu Gao, Grant Ayers, and Christos Kozyrakis. 2015. Practical near-data processing for in-memory analytics frameworks. In *2015 International Conference on Parallel Architecture and Compilation (PACT)*. IEEE, 113–124.

[79] Mingyu Gao and Christos Kozyrakis. 2016. HRL: Efficient and flexible reconfigurable logic for near-data processing. In *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. Ieee, 126–137.

[80] Mingyu Gao, Jing Pu, Xuan Yang, Mark Horowitz, and Christos Kozyrakis. 2017. Tetris: Scalable and efficient neural network acceleration with 3d memory. In *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*. 751–764.

[81] Sahar Ghoflsaz Ghinani, Jingyao Zhang, and Elaheh Sadredini. 2025. Enabling Low-Cost Secure Computing on Untrusted In-Memory Architectures. *arXiv preprint arXiv:2501.17292* (2025).

[82] Christina Giannoula, Ivan Fernandez, Juan Gómez-Luna, Nectarios Koziris, Georgios Goumas, and Onur Mutlu. 2022. Towards efficient sparse matrix vector multiplication on real processing-in-memory architectures. *ACM SIGMETRICS Performance Evaluation Review* 50, 1 (2022), 33–34.

[83] Christina Giannoula, Ivan Fernandez, Juan Gómez Luna, Nectarios Koziris, Georgios Goumas, and Onur Mutlu. 2022. Sparsep: Towards efficient sparse matrix vector multiplication on real processing-in-memory architectures. *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 6, 1 (2022), 1–49.

[84] Christina Giannoula, Nandita Vijaykumar, Nikela Papadopoulou, Vasileios Karakostas, Ivan Fernandez, Juan Gómez-Luna, Lois Orosa, Nectarios Koziris, Georgios Goumas, and Onur Mutlu. 2021. Syncron: Efficient synchronization support for near-data-processing architectures. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 263–276.

[85] Christina Giannoula, Peiming Yang, Ivan Fernandez, Jiacheng Yang, Sankeerth Durvasula, Yu Xin Li, Mohammad Sadrosadati, Juan Gomez Luna, Onur Mutlu, and Gennady Pekhimenko. 2024. PyGim: An Efficient Graph Neural Network Library for Real Processing-In-Memory Architectures. *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 8, 3 (2024), 1–36.

[86] Alvin Oliver Glova, Itir Akgun, Shuangchen Li, Xing Hu, and Yuan Xie. 2019. Near-data acceleration of privacy-preserving biomarker search with 3D-stacked memory. In *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 800–805.

[87] Kailash Gogineni, Sai Santosh Dayapule, Juan Gómez-Luna, Karthikeya Gogineni, Peng Wei, Tian Lan, Mohammad Sadrosadati, Onur Mutlu, and Guru Venkataramani. 2024. Swiftrl: Towards efficient reinforcement learning on real processing-in-memory systems. In *2024 IEEE International Symposium on Performance Analysis of Systems and Software*

(ISPASS). IEEE, 217–229.

[88] Juan Gómez-Luna, Izzat El Hajj, Ivan Fernandez, Christina Giannoula, Geraldo F Oliveira, and Onur Mutlu. 2021. Benchmarking memory-centric computing systems: Analysis of real processing-in-memory hardware. In *2021 12th International Green and Sustainable Computing Conference (IGSC)*. IEEE, 1–7.

[89] Juan Gómez-Luna, Izzat El Hajj, Ivan Fernandez, Christina Giannoula, Geraldo F Oliveira, and Onur Mutlu. 2022. Benchmarking a new paradigm: Experimental analysis and characterization of a real processing-in-memory system. *IEEE Access* 10 (2022), 52565–52608.

[90] Juan Gómez-Luna, Yuxin Guo, Sylvan Brocard, Julien Legriel, Remy Cimadomo, Geraldo F Oliveira, Gagandeep Singh, and Onur Mutlu. 2022. Machine learning training on a real processing-in-memory system. In *2022 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 292–295.

[91] Peng Gu, Xinfeng Xie, Yufei Ding, Guoyang Chen, Weifeng Zhang, Dimin Niu, and Yuan Xie. 2020. iPIM: Programmable in-memory image processing accelerator using near-bank architecture. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 804–817.

[92] Peng Gu, Xinfeng Xie, Shuangchen Li, Dimin Niu, Hongzhong Zheng, Krishna T Malladi, and Yuan Xie. 2020. DLUX: A LUT-based near-bank accelerator for data center deep learning training workloads. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 40, 8 (2020), 1586–1599.

[93] Yufeng Gu, Alireza Khadem, Sumanth Umesh, Ning Liang, Xavier Servot, Onur Mutlu, Ravi Iyer, and Reetuparna Das. 2025. 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, Volume 2*. 862–881.

[94] Harshita Gupta, Mayank Kabra, Juan Gómez-Luna, Konstantinos Kanellopoulos, and Onur Mutlu. 2023. Evaluating homomorphic operations on a real-world processing-in-memory system. In *2023 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 211–215.

[95] Zvika Guz, Manu Awasthi, Vijay Balakrishnan, Mrinmoy Ghosh, Anahita Shayesteh, Tameesh Suri, and Samsung Semiconductor. 2014. Real-time analytics as the killer application for processing-in-memory. *Near Data Processing (WoNDP)* (2014), 10–2.

[96] Ramyad Hadidi, Lifeng Nai, Hyojong Kim, and Hyesoon Kim. 2017. Cairo: A compiler-assisted technique for enabling instruction-level offloading of processing-in-memory. *ACM Transactions on Architecture and Code Optimization (TACO)* 14, 4 (2017), 1–25.

[97] 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*. 329–345.

[98] Hyungkyu Ham, Jeongmin Hong, Geonwoo Park, Yunseon Shin, Okkyun Woo, Wonhyuk Yang, Jinhoon Bae, Eunhyeok Park, Hyojin Sung, Euicheol Lim, et al. 2024. Low-overhead general-purpose near-data processing in cxl memory expanders. In *2024 57th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 594–611.

[99] Lei He, Cheng Liu, Ying Wang, Shengwen Liang, Huawei Li, and Xiaowei Li. 2021. Gcim: a near-data processing accelerator for graph construction. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 205–210.

[100] Mingxuan He, Choungki Song, Ilkon Kim, Chunseok Jeong, Seho Kim, Il Park, Mithuna Thottethodi, and TN 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)*. IEEE, 372–385.

[101] Yintao He, Haiyu Mao, Christina Giannoula, Mohammad Sadrosadati, Juan Gómez-Luna, Huawei Li, Xiaowei Li, Ying Wang, and Onur Mutlu. 2025. Papi: Exploiting dynamic parallelism in large language model decoding with a processing-in-memory-enabled computing system. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*. 766–782.

[102] 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*. 722–737.

[103] Jaehoon Heo, Yongwon Shin, Sangjin Choi, Sungwoong Yune, Jung-Hoon Kim, Hyojin Sung, Youngjin Kwon, and Joo-Young Kim. 2023. Primo: A full-stack processing-in-dram emulation framework for machine learning workloads. In *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. IEEE, 1–9.

[104] Jaeyoung Heo and Sungjoo Yoo. 2024. NeRF-PIM: PIM Hardware-Software Co-Design of Neural Rendering Networks. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 43, 11 (2024), 3900–3912.

[105] Byungchul Hong, Yeonju Ro, and John Kim. 2018. Multi-dimensional parallel training of winograd layer on memory-centric architecture. In *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 682–695.

[106] Seongyon Hong, Wooyoung Jo, Sangjin Kim, Sangyeob Kim, Kyomin Sohn, and Hoi-Jun Yoo. 2024. Dyamond: A 1T1C DRAM In-memory Computing Accelerator with Compact MAC-SIMD and Adaptive Column Addition Dataflow. In *2024 IEEE Symposium on VLSI Technology and Circuits (VLSI Technology and Circuits)*. IEEE, 1–2.

[107] Chi-Tse Huang, Cheng-Yang Chang, Yu-Chuan Chuang, and An-Yeu Andy Wu. 2023. BWA-NIMC: Budget-based Workload Allocation for Hybrid Near/In-Memory-Computing. In *2023 60th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.

[108] Jiayi Huang, Pritam Majumder, Sungkeun Kim, Troy Fulton, Ramprakash Reddy Puli, Ki Hwan Yum, and Eun Jung Kim. 2021. Computing en-route for near-data processing. *IEEE Trans. Comput.* 70, 6 (2021), 906–921.

[109] Jiayi Huang, Ramprakash Reddy Puli, Pritam Majumder, Sungkeun Kim, Rahul Boyapati, Ki Hwan Yum, and Eun Jung Kim. 2019. Active-routing: Compute on the way for near-data processing. In *2019 IEEE International symposium on high performance computer architecture (HPCA)*. IEEE, 674–686.

[110] Yu Huang, Long Zheng, Haifeng Liu, Zhuoran Zhou, Dan Chen, Pengcheng Yao, Qinggang Wang, Xiaofei Liao, and Hai Jin. 2023. MeG 2: In-Memory Acceleration for Genome Graphs Analysis. In *2023 60th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.

[111] Wenqin Huangfu, Xueqi Li, Shuangchen Li, Xing Hu, Peng Gu, and Yuan Xie. 2019. Medal: Scalable dimm based near data processing accelerator for dna seeding algorithm. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*. 587–599.

[112] Wenqin Huangfu, Krishna T Malladi, Andrew Chang, and Yuan Xie. 2022. Beacon: Scalable near-data-processing accelerators for genome analysis near memory pool with the cxl support. In *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 727–743.

[113] Wenqin Huangfu, Krishna T Malladi, Shuangchen Li, Peng Gu, and Yuan Xie. 2020. NEST: DIMM based near-data-processing accelerator for k-mer counting, in 2020 IEEE. In *ACM International Conference On Computer Aided Design (ICCAD)*. 1–9.

[114] Bongjoon Hyun, Taehun Kim, Dongjae Lee, and Minsoo Rhu. 2024. Pathfinding future pim architectures by demystifying a commercial pim technology. In *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 263–279.

[115] Mohamed A Ibrahim, Shaizeen Aga, Ada Li, Suchita Pati, and Mahzabeen Islam. 2024. JIT-Q: Just-in-time Quantization with Processing-In-Memory for Efficient ML Training. *Proceedings of Machine Learning and Systems* 6 (2024), 46–59.

[116] Bruce Jacob, David Wang, and Spencer Ng. 2010. *Memory systems: cache, DRAM, disk.* Morgan Kaufmann.

[117] Junhyeok Jang, Hanjin Choi, Hanyeoreum Bae, Seungjun Lee, Miryeong Kwon, and Myoungsoo Jung. 2023. {CXL-ANNS}:{Software-Hardware} collaborative memory disaggregation and computation for {Billion-Scale} approximate nearest neighbor search. In *2023 USENIX Annual Technical Conference (USENIX ATC 23)*. 585–600.

[118] Jaeyoung Jang, Jun Heo, Yejin Lee, Jaeyeon Won, Seonghak Kim, Sung Jun Jung, Hakbeom Jang, Tae Jun Ham, and Jae W Lee. 2019. Charon: Specialized near-memory processing architecture for clearing dead objects in memory. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*. 726–739.

[119] Je-Woo Jang, Junyong Oh, Youngbae Kong, Jae-Youn Hong, Sung-Hyuk Cho, Jeongyeol Lee, Hoeseok Yang, and Joon-Sung Yang. 2025. Accelerating Retrieval Augmented Language Model via PIM and PNM Integration. In *Proceedings of the 58th IEEE/ACM International Symposium on Microarchitecture®*. 246–262.

[120] Shengyi Ji, Chubo Liu, Yan Ding, Qing Liao, and Zhuo Tang. 2024. A Real-time Execution System of Multimodal Transformer through PIM-GPU Collaboration. In *Proceedings of the 61st ACM/IEEE Design Automation Conference*. 1–6.

[121] Hongyang Jia, Murat Ozatay, Yinqi Tang, Hossein Valavi, Rakshit Pathak, Jinseok Lee, and Naveen Verma. 2021. 15.1 a programmable neural-network inference accelerator based on scalable in-memory computing. In *2021 IEEE International Solid-State Circuits Conference (ISSCC)*, Vol. 64. IEEE, 236–238.

[122] Qingcai Jiang, Shaojie Tan, Junshi Chen, and Hong An. 2024. A 3 PIM: An automated, analytic and accurate processing-in-memory offloader. In *2024 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 1–6.

[123] Qingcai Jiang, Buxin Tu, and Hong An. 2025. NDPage: Efficient Address Translation for Near-Data Processing Architectures via Tailored Page Table. In *2025 Design, Automation & Test in Europe Conference (DATE)*. IEEE, 1–7.

[124] Hai Jin, Dan Chen, Long Zheng, Yu Huang, Pengcheng Yao, Jin Zhao, Xiaofei Liao, and Wenbin Jiang. 2023. Accelerating graph convolutional networks through a PIM-accelerated approach. *IEEE Trans. Comput.* 72, 9 (2023), 2628–2640.

[125] Gilbert Jonatan, Haeyoon Cho, Hyojun Son, Xiangyu Wu, Neal Livesay, Evelio Mora, Kaustubh Shivdikar, José L Abellán, Ajay Joshi, David Kaeli, et al. 2024. Scalability limitations of processing-in-memory using real system evaluations. *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 8, 1 (2024), 1–28.

[126] Mayank Kabra, Rakesh Nadig, Harshita Gupta, Rahul Bera, Manos Frouzakis, Vamanan Arulchelvan, Yu Liang, Haiyu Mao, Mohammad Sadrosadati, and Onur Mutlu. 2025. CIPHERMATCH: Accelerating Homomorphic Encryption-Based String Matching via Memory-Efficient Data Packing and In-Flash Processing. In *Proceedings of the 30th ACM*

International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2. 111–130.

[127] Hongju Kal, Chanyoung Yoo, and Won Woo Ro. 2023. Aespa: Asynchronous execution scheme to exploit bank-level parallelism of processing-in-memory. In *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*. 815–827.

[128] Hongbo Kang, Phillip B Gibbons, Guy E Blelloch, Laxman Dhulipala, Yan Gu, and Charles McGuffey. 2021. The processing-in-memory model. In *Proceedings of the 33rd ACM Symposium on Parallelism in Algorithms and Architectures*. 295–306.

[129] Hongbo Kang, Yiwei Zhao, Guy E Blelloch, Laxman Dhulipala, Yan Gu, Charles McGuffey, and Phillip B Gibbons. 2025. PIM-tree: A Skew-resistant Index for Processing-in-Memory: H. Kang et al. *The VLDB Journal* 34, 6 (2025), 66.

[130] Luyi Kang, Yuqi Xue, Weiwei Jia, Xiaohao Wang, Jongryool Kim, Changhwan Youn, Myeong Joon Kang, Hyung Jin Lim, Bruce Jacob, and Jian Huang. 2021. Iceclave: A trusted execution environment for in-storage computing. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*. 199–211.

[131] Shinhaeng Kang, Sukhan Lee, Byeongho Kim, Hweesoo Kim, Kyomin Sohn, Nam Sung Kim, and Eojin Lee. 2022. An fpga-based rnn-t inference accelerator with pim-hbm. In *Proceedings of the 2022 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. 146–152.

[132] Liu Ke, Udit Gupta, Benjamin Youngjae Cho, David Brooks, Vikas Chandra, Utku Diril, Amin Firoozshahian, Kim Hazelwood, Bill Jia, Hsien-Hsin S Lee, et al. 2020. Recnmp: Accelerating personalized recommendation with near-memory processing. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 790–803.

[133] Liu Ke, Xuan Zhang, Jinin So, Jong-Geon Lee, Shin-Haeng Kang, Sukhan Lee, Songyi Han, YeonGon Cho, Jin Hyun Kim, Yongsuk Kwon, et al. 2021. Near-memory processing in action: Accelerating personalized recommendation with axdimm. *IEEE Micro* 42, 1 (2021), 116–127.

[134] Tiago R Kepe, Eduardo C de Almeida, and Marco AZ Alves. 2019. Database processing-in-memory: An experimental study. *Proceedings of the VLDB Endowment* 13, 3 (2019), 334–347.

[135] Bokyung Kim. 2025. A DRAM-Based Processing-in-Memory Accelerator for Privacy-Protecting Machine Learning. In *2025 Design, Automation & Test in Europe Conference (DATE)*. IEEE, 1–2.

[136] Byeongho Kim, Jongwook Chung, Eojin Lee, Wonkyung Jung, Sunjung Lee, Jaewan Choi, Jaehyun Park, Minbok Wi, Sukhan Lee, and Jung Ho Ahn. 2020. MViD: Sparse matrix-vector multiplication in mobile dram for accelerating recurrent neural networks. *IEEE Trans. Comput.* 69, 7 (2020), 955–967.

[137] Donghyeon Kim, Taehoon Kim, Inyong Hwang, Taehyeong Park, Hanjun Kim, Youngsok Kim, and Yongjun Park. 2023. Virtual pim: Resource-aware dynamic dpu allocation and workload scheduling framework for multi-dpu pim architecture. In *2023 32nd International Conference on Parallel Architectures and Compilation Techniques (PACT)*. IEEE, 112–123.

[138] Duckhwan Kim, Jaeha Kung, Sek Chai, Sudhakar Yalamanchili, and Saibal Mukhopadhyay. 2016. Neurocube: A programmable digital neuromorphic architecture with high-density 3D memory. *ACM SIGARCH Computer Architecture News* 44, 3 (2016), 380–392.

[139] Gwangsun Kim, Niladrish Chatterjee, Mike O'Connor, and Kevin Hsieh. 2017. Toward standardized near-data processing with unrestricted data placement for GPUs. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–12.

[140] Heesu Kim, Hanmin Park, Taehyun Kim, Kwanheum Cho, Eojin Lee, Soojung Ryu, Hyuk-Jae Lee, Kiyoung Choi, and Jinho Lee. 2021. GradPIM: A practical processing-in-DRAM architecture for gradient descent. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 249–262.

[141] Hyeonuk Kim, Jaehyeong Sim, Yeongjae Choi, and Lee-Sup Kim. 2019. NAND-Net: Minimizing computational complexity of in-memory processing for binary neural networks. In *2019 IEEE international symposium on high performance computer architecture (HPCA)*. IEEE, 661–673.

[142] Hyoungjoo Kim, Yiwei Zhao, Andrew Pavlo, and Phillip B Gibbons. 2025. No Cap, This Memory Slaps: Breaking Through the Memory Wall of Transactional Database Systems with Processing-in-Memory. *Proceedings of the VLDB Endowment* 18, 11 (2025), 4241–4254.

[143] Jeehyun Kim, Donghyeon Kim, Seokwon Kang, Bongjoon Hyun, Inho Lee, and Yongjun Park. 2025. PIM-CCA: An Efficient PIM Architecture with Optimized Integration of Configurable Functional Units. In *Proceedings of the 58th IEEE/ACM International Symposium on Microarchitecture®*. 63–77.

[144] Jongmin Kim, Sungmin Yun, Hyesung Ji, Wonseok Choi, Sangpyo Kim, and Jung Ho Ahn. 2025. Anaheim: Architecture and Algorithms for Processing Fully Homomorphic Encryption in Memory. In *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 1158–1173.

[145] Jin Hyun Kim, Shin-haeng Kang, Sukhan Lee, Hyeonsu Kim, Woongjae Song, Yuhwan Ro, Seungwon Lee, David Wang, Hyunsung Shin, Bengseng Phuah, et al. 2021. Aquabolt-XL: Samsung HBM2-PIM with in-memory processing

for ML accelerators and beyond. In *2021 IEEE Hot Chips 33 Symposium (HCS)*. IEEE, 1–26.

[146] Jin Hyun Kim, Yuhwan Ro, Jinin So, Sukhan Lee, Shin-haeng Kang, YeonGon Cho, Hyeonsu Kim, Byeongho Kim, Kyungsoo Kim, Sangsoo Park, et al. 2023. Samsung pim/pnm for transfmer based ai: Energy efficiency on pim/pnm cluster. In *2023 IEEE Hot Chips 35 Symposium (HCS)*. IEEE Computer Society, 1–31.

[147] Jeremie S Kim, Damla Senol Cali, Hongyi Xin, Donghyuk Lee, Saugata Ghose, Mohammed Alser, Hasan Hassan, Oguz Ergin, Can Alkan, and Onur Mutlu. 2018. GRIM-Filter: Fast seed location filtering in DNA read mapping using processing-in-memory technologies. *BMC genomics* 19, Suppl 2 (2018), 89.

[148] Nam Sung Kim and Pankaj Mehra. 2019. Practical near-data processing to evolve memory and storage devices into mainstream heterogeneous computing systems. In *Proceedings of the 56th Annual Design Automation Conference 2019*. 1–4.

[149] Taehwan Kim, Yunki Han, Seohye Ha, Jiwan Kim, and Lee-Sup Kim. 2025. EOD: Enabling Low Latency GNN Inference via Near-Memory Concatenate Aggregation. In *Proceedings of the 52nd Annual International Symposium on Computer Architecture*. 1125–1139.

[150] Wonung Kim, Yubin Lee, Yoonsung Kim, Jinwoo Hwang, Seongryong Oh, Jiyong Jung, Aziz Huseynov, Woong Gyu Park, Chang Hyun Park, Divya Mahajan, et al. 2025. Pimba: A Processing-in-Memory Acceleration for Post-Transformer Large Language Model Serving. In *Proceedings of the 58th IEEE/ACM International Symposium on Microarchitecture®*. 292–307.

[151] Yoongu Kim, Weikun Yang, and Onur Mutlu. 2015. Ramulator: A fast and extensible DRAM simulator. *IEEE Computer architecture letters* 15, 1 (2015), 45–49.

[152] Seoyoung Ko, Hyunjeong Shim, Wanju Doh, Sungmin Yun, Jinin So, Yongsuk Kwon, Sang-Soo Park, Si-Dong Roh, Minyong Yoon, Taeksang Song, et al. 2025. COSMOS: A CXL-Based Full In-Memory System for Approximate Nearest Neighbor Search. *IEEE Computer Architecture Letters* (2025).

[153] Gokul Krishnan, A Alper Goksoy, Sumit K Mandal, Zhenyu Wang, Chaitali Chakrabarti, Jae-sun Seo, Umit Y Ogras, and Yu Cao. 2022. Big-little chiplets for in-memory acceleration of dnns: A scalable heterogeneous architecture. In *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*. 1–9.

[154] Daehan Kwon, Seongju Lee, Kyuyoung Kim, Sanghoon Oh, Joonhong Park, Gi-Moon Hong, Dongyoon Ka, Kyudong Hwang, Jeongje Park, Kyeongpil Kang, et al. 2022. A 1ynm 1.25 v 8gb 16gb/s/pin gddr6-based accelerator-in-memory supporting 1tflops mac operation and various activation functions for deep learning application. *IEEE Journal of Solid-State Circuits* 58, 1 (2022), 291–302.

[155] Hyucksung Kwon, Kyungmo Koo, Janghyeon Kim, Woongkyu Lee, Minjae Lee, Hyungdeok Lee, Yousub Jung, Jaehan Park, Yosub Song, Byeongsu Yang, et al. 2024. Lol-pim: Long-context llm decoding with scalable dram-pim system. *arXiv preprint arXiv:2412.20166* (2024).

[156] Youngeun Kwon, Yunjae Lee, and Minsoo Rhu. 2019. Tensordimm: A practical near-memory processing architecture for embeddings and tensor operations in deep learning. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*. 740–753.

[157] Yongkee Kwon, Kornijcuk Vladimir, Nahsung Kim, Woojae Shin, Jongsoon Won, Minkyu Lee, Hyunha Joo, Haerang Choi, Guhyun Kim, Byeongju An, et al. 2022. System architecture and software stack for GDDR6-AiM. In *2022 IEEE Hot Chips 34 Symposium (HCS)*. IEEE, 1–25.

[158] Young-Cheon Kwon, Suk Han Lee, Jaehoon Lee, Sang-Hyuk Kwon, Je Min Ryu, Jong-Pil Son, O Seongil, Hak-Soo Yu, Haesuk Lee, Soo Young Kim, et al. 2021. 25.4 a 20nm 6gb function-in-memory dram, based on hbm2 with a 1.2 tflops programmable computing unit using bank-level parallelism, for machine learning applications. In *2021 IEEE International Solid-State Circuits Conference (ISSCC)*, Vol. 64. IEEE, 350–352.

[159] Dominique Lavenier, Remy Cimadomo, and Romaric Jodin. 2020. Variant calling parallelization on processor-in-memory architecture. In *2020 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*. IEEE, 204–207.

[160] Dongjae Lee, Bongjoon Hyun, Taehun Kim, and Minsoo Rhu. 2024. PIM-MMU: A Memory Management Unit for Accelerating Data Transfers in Commercial PIM Systems. In *2024 57th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 627–642.

[161] Donghun Lee, Jinin So, Minseon Ahn, Jong-Geon Lee, Jungmin Kim, Jeonghyeon Cho, Rebholz Oliver, Vishnu Charan Thummala, Ravi shankar JV, Sachin Suresh Upadhya, et al. 2022. Improving in-memory database operations with acceleration dimm (axdimm). In *Proceedings of the 18th International Workshop on Data Management on New Hardware*. 1–9.

[162] Hyojung Lee, Daehyeon Baek, Jimyoung Son, Jieun Choi, Kihyo Moon, and Minsung Jang. 2025. PAISE: PIM-Accelerated Inference Scheduling Engine for Transformer-based LLM. In *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 1707–1719.

[163] Hyungdeok Lee, Guhyun Kim, Dayeon Yun, Ilkon Kim, Yongkee Kwon, and Euicheol Lim. 2024. Cost-effective llm accelerator using processing in memory technology. In *2024 IEEE Symposium on VLSI Technology and Circuits (VLSI Technology and Circuits)*. IEEE, 1–2.

[164] Minkyu Lee, Sang-Seol Lee, Kyungho Kim, Eunchong Lee, and Sung-Joon Jang. 2024. HAIL-DIMM: Host Access Interleaved with Near-Data Processing on DIMM-based Memory System. In *Proceedings of the 61st ACM/IEEE Design Automation Conference*. 1–6.

[165] Sukhan Lee, Shin-haeng Kang, Jaehoon Lee, Hyeonsu Kim, Eojin Lee, Seungwoo Seo, Hosang Yoon, Seungwon Lee, Kyounghwan Lim, Hyunsung Shin, et al. 2021. Hardware architecture and software stack for PIM based on commercial DRAM technology: Industrial product. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 43–56.

[166] Seongju Lee, Kyuyoung Kim, Sanghoon Oh, Joonhong Park, Gimoon Hong, Dongyoon Ka, Kyudong Hwang, Jeongje Park, Kyeongpil Kang, Jungyeon Kim, et al. 2022. A 1ynm 1.25 V 8Gb, 16Gb/s/pin GDDR6-based accelerator-in-memory supporting 1TFLOPS MAC operation and various activation functions for deep-learning applications. In *2022 IEEE International Solid-State Circuits Conference (ISSCC)*, Vol. 65. IEEE, 1–3.

[167] Sanghoon Lee, Jongho Park, Minho Ha, Byung Il Koh, Kyoung Park, and Yeseong Kim. 2023. Sidekick: Near Data Processing for Clustering Enhanced by Automatic Memory Disaggregation. In *2023 60th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.

[168] Vincent T Lee, Amrita Mazumdar, Carlo C del Mundo, Armin Alaghi, Luis Ceze, and Mark Oskin. 2018. Application codesign of near-data processing for similarity search. In *2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 896–907.

[169] Young Sik Lee and Tae Hee Han. 2021. Task parallelism-aware deep neural network scheduling on multiple hybrid memory cube-based processing-in-memory. *IEEE Access* 9 (2021), 68561–68572.

[170] Marzieh Lenjani, Alif Ahmed, Mircea Stan, and Kevin Skadron. 2022. Gearbox: A case for supporting accumulation dispatching and hybrid partitioning in PIM-based accelerators. In *Proceedings of the 49th Annual International Symposium on Computer Architecture*. 218–230.

[171] Cong Li, Yihan Yin, Xintong Wu, Jingchen Zhu, Zhutianya Gao, Dimin Niu, Qiang Wu, Xin Si, Yuan Xie, Chen Zhang, et al. 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*. 194–210.

[172] Cong Li, Zhe Zhou, Xingchen Li, Guangyu Sun, and Dimin Niu. 2023. Nmexplorer: An efficient exploration framework for dimm-based near-memory tensor reduction. In *2023 60th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.

[173] Cong Li, Zhe Zhou, Yang Wang, Fan Yang, Ting Cao, Mao Yang, Yun Liang, and Guangyu Sun. 2024. Pim-dl: Expanding the applicability of commodity dram-pims for deep learning via algorithm-system co-optimization. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*. 879–896.

[174] Cong Li, Zhe Zhou, Size Zheng, Jiaxi Zhang, Yun Liang, and Guangyu Sun. 2024. Specpim: Accelerating speculative inference on pim-enabled system via architecture-dataflow co-exploration. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*. 950–965.

[175] 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)*. IEEE, 696–709.

[176] Shuangchen Li, Dimin Niu, Krishna T Malladi, Hongzhong Zheng, Bob Brennan, and Yuan Xie. 2017. Drisa: A dram-based reconfigurable in-situ accelerator. In *Proceedings of the 50th annual ieee/acm international symposium on microarchitecture*. 288–301.

[177] Shiyu Li, Yitu Wang, Edward Hanson, Andrew Chang, Yang Seok Ki, Hai Li, and Yiran Chen. 2024. Ndrec: A near-data processing system for training large-scale recommendation models. *IEEE Trans. Comput.* 73, 5 (2024), 1248–1261.

[178] Wen Li, Ying Wang, Huawei Li, and Xiaowei Li. 2019. P3M: a PIM-based neural network model protection scheme for deep learning accelerator. In *Proceedings of the 24th Asia and South Pacific Design Automation Conference*. 633–638.

[179] Yiwei Li, Yuxin Jin, Boyu Tian, Huanchen Zhang, and Mingyu Gao. 2025. ANSMET: Approximate Nearest Neighbor Search with Near-Memory Processing and Hybrid Early Termination. In *Proceedings of the 52nd Annual International Symposium on Computer Architecture*. 1093–1107.

[180] Yiwei Li, Boyu Tian, Yi Ren, and Mingyu Gao. 2024. Stream-Based Data Placement for Near-Data Processing with Extended Memory. In *2024 57th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 1648–1662.

[181] Zerun Li, Xiaoming Chen, and Yinhe Han. 2022. Optimal data allocation for graph processing in processing-in-memory systems. In *2022 27th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 238–243.

[182] Zerun Li, Xiaoming Chen, and Yinhe Han. 2024. TMiner: A Vertex-Based Task Scheduling Architecture for Graph Pattern Mining. In *2024 57th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 1295–1308.

[183] Shengwen Liang, Ziming Yuan, Ying Wang, Dawen Xu, Huawei Li, and Xiaowei Li. 2024. HyQA: Hybrid Near-Data Processing Platform for Embedding Based Question Answering System. In *2024 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 1–6.

[184] Chaemin Lim, Suhyun Lee, Jinwoo Choi, Jounghoo Lee, Seongyeon Park, Hanjun Kim, Jinho Lee, and Youngsok Kim. 2023. Design and analysis of a processing-in-dimm join algorithm: A case study with upmem dimms. *Proceedings of the ACM on Management of Data* 1, 2 (2023), 1–27.

[185] Junfeng Lin, Huanyu Qu, Songchen Ma, Xinglong Ji, Hongyi Li, Xiaochuan Li, Chenhang Song, and Weihao Zhang. 2023. SongC: A compiler for hybrid near-memory and in-memory many-core architecture. *IEEE Trans. Comput.* (2023).

[186] Kuan-Chih Lin, Hao Zuo, Hsiang-Yu Wang, Yuan-Ping Huang, Ci-Hao Wu, Yan-Cheng Guo, Shyh-Jye Jou, Tuo-Hung Hou, and Tian-Sheuan Chang. 2024. A multi-bit near-rram based computing macro with highly computing parallelism for cnn application. In *2024 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 1–6.

[187] Chaoqiang Liu, Haifeng Liu, Dan Chen, Yu Huang, Yi Zhang, Wenjing Xiao, Xiaofei Liao, and Hai Jin. 2025. HeterRAG: Heterogeneous Processing-in-Memory Acceleration for Retrieval-augmented Generation. In *Proceedings of the 52nd Annual International Symposium on Computer Architecture*. 884–898.

[188] Haifeng Liu, Long Zheng, Yu Huang, Haoyan Huang, Xiaofei Liao, and Jin Hai. 2024. Towards Redundancy-Free Recommendation Model Training via Reusable-aware Near-Memory Processing. In *Proceedings of the 61st ACM/IEEE Design Automation Conference*. 1–6.

[189] Haifeng Liu, Long Zheng, Yu Huang, Chaoqiang Liu, Xiangyu Ye, Jingrui Yuan, Xiaofei Liao, Hai Jin, and Jingling Xue. 2023. Accelerating personalized recommendation with cross-level near-memory processing. In *Proceedings of the 50th Annual International Symposium on Computer Architecture*. 1–13.

[190] Haifeng Liu, Long Zheng, Yu Huang, Jingyi Zhou, Chaoqiang Liu, Runze Wang, Xiaofei Liao, Hai Jin, and Jingling Xue. 2024. Enabling efficient large recommendation model training with near cxl memory processing. In *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 382–395.

[191] Jiawen Liu, Hengyu Zhao, Matheus A Ogleari, Dong Li, and Jishen Zhao. 2018. Processing-in-memory for energy-efficient neural network training: A heterogeneous approach. In *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 655–668.

[192] 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*. 867–883.

[193] Liu Liu, Jilan Lin, Zheng Qu, Yufei Ding, and Yuan Xie. 2021. Enmc: Extreme near-memory classification via approximate screening. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*. 1309–1322.

[194] Lian Liu, Shixin Zhao, Bing Li, Haimeng Ren, Zhaohui Xu, Mengdi Wang, Xiaowei Li, Yinhe Han, and Ying Wang. 2025. Make llm inference affordable to everyone: Augmenting gpu memory with ndp-dimm. In *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 1751–1765.

[195] André Lopes, Daniel Castro, and Paolo Romano. 2024. PIM-STM: Software transactional memory for processing-in-memory systems. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*. 897–911.

[196] Shih-Lien Lu, Jeonghyun Woo, and Prashant J Nair. 2025. Counterpoint: One-Hot Counting for PRAC-Based RowHammer Mitigation. *DRAMSec* (2025).

[197] Ruoyan Ma, Shengan Zheng, Guifeng Wang, Jin Pu, Yifan Hua, Wentao Wang, and Linpeng Huang. 2024. Accelerating regular path queries over graph database with processing-in-memory. In *Proceedings of the 61st ACM/IEEE Design Automation Conference*. 1–6.

[198] Andrew McCrabb, Aymen Ahmed, and Valeria Bertacco. 2023. Acre: Accelerating random forests for explainability. In *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*. 1016–1028.

[199] Andrew McCrabb, Ivris Raymond, and Valeria Bertacco. 2025. GLEAM: Graph-Based Learning Through Efficient Aggregation in Memory. In *2025 Design, Automation & Test in Europe Conference (DATE)*. IEEE, 1–7.

[200] Rafael Medina, Giovanni Ansaloni, Marina Zapater, Alexandre Levisse, Saeideh Alinezhad Chamazcoti, Timon Evenblij, Dwaipayan Biswas, Francky Catthoor, and David Atienza. 2024. Bank on compute-near-memory: design space exploration of processing-near-bank architectures. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 43, 11 (2024), 4117–4129.

[201] Chuhan Min, Jiachen Mao, Hai Li, and Yiran Chen. 2019. NeuralHMC: An efficient HMC-based accelerator for deep neural networks. In *Proceedings of the 24th Asia and South Pacific Design Automation Conference*. 394–399.

[202] Sergiu Mosanu, Mohammad Nazmus Sakib, Tommy Tracy, Ersin Cukurtas, Alif Ahmed, Preslav Ivanov, Samira Khan, Kevin Skadron, and Mircea Stan. 2022. PiMulator: A fast and flexible processing-in-memory emulation platform. In *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 1473–1478.

[203] Onur Mutlu, Saugata Ghose, Juan Gómez-Luna, and Rachata Ausavarungnirun. 2019. Enabling practical processing in and near memory for data-intensive computing. In *Proceedings of the 56th Annual Design Automation Conference 2019*. 1–4.

[204] Anirban Nag and Rajeev Balasubramonian. 2021. OrderLight: Lightweight memory-ordering primitive for efficient fine-grained PIM computations. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*. 298–310.

[205] Lifeng Nai, Ramyad Hadidi, Jaewoong Sim, Hyojong Kim, Pranith Kumar, and Hyesoon Kim. 2017. Graphpim: Enabling instruction-level pim offloading in graph computing frameworks. In *2017 IEEE International symposium on high performance computer architecture (HPCA)*. IEEE, 457–468.

[206] Kevin Nam, Heonhui Jung, Hyunyoung Oh, and Yunheung Paek. 2025. Affinity-based Optimizations for TFHE on Processing-in-DRAM. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*. 16–31.

[207] Joel Nider, Craig Mustard, Andrada Zoltan, John Ramsden, Larry Liu, Jacob Grossbard, Mohammad Dashti, Romaric Jodin, Alexandre Ghiti, Jordi Chauzi, et al. 2021. A case study of {Processing-in-Memory} in {off-the-Shelf} systems. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*. 117–130.

[208] Dimitra Nikitopoulou, Dimosthenis Masouros, Sotirios Xydis, and Dimitrios Soudris. 2021. Performance analysis and auto-tuning for spark in-memory analytics. In *2021 Design, automation & test in Europe conference & exhibition (DATE)*. IEEE, 76–81.

[209] Dimin Niu, Shuangchen Li, Yuhao Wang, Wei Han, Zhe Zhang, Yijin Guan, Tianchan Guan, Fei Sun, Fei Xue, Lide Duan, et al. 2022. 184QPS/W 64Mb/mm 2 3D logic-to-DRAM hybrid bonding with process-near-memory engine for recommendation system. In *2022 IEEE International Solid-State Circuits Conference (ISSCC)*, Vol. 65. IEEE, 1–3.

[210] Si Ung Noh, Junguk Hong, Chaemin Lim, Seongyeon Park, Jeehyun Kim, Hanjun Kim, Youngsok Kim, and Jinho Lee. 2024. Pid-comm: A fast and flexible collective communication framework for commodity processing-in-dimm devices. In *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 245–260.

[211] Geraldo F Oliveira, Juan Gómez-Luna, Mohammad Sadrosadati, Yuxin Guo, and Onur Mutlu. 2023. Transpimlib: Efficient transcendental functions for processing-in-memory systems. In *2023 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 235–247.

[212] 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)*. IEEE, 186–203.

[213] Geraldo F Oliveira, Paulo C Santos, Marco AZ Alves, and Luigi Carro. 2017. A generic processing in memory cycle accurate simulator under hybrid memory cube architecture. In *2017 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS)*. IEEE, 54–61.

[214] Marcelo Orenes-Vera, Esin Tureci, David Wentzlaff, and Margaret Martonosi. 2023. Dalorex: A data-local program execution and architecture for memory-bound applications. In *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 718–730.

[215] Cristobal Ortega, Yann Falevoz, and Renaud Ayrignac. 2024. Pim-ai: A novel architecture for high-efficiency llm inference. *arXiv preprint arXiv:2411.17309* (2024).

[216] Yue Pan, Zihan Xia, Po-Kai Hsu, Lanxiang Hu, Hyungyo Kim, Janak Sharda, Minxuan Zhou, Nam Sung Kim, Shimeng Yu, Tajana Rosing, et al. 2025. 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®*. 1–17.

[217] Yue Pan, Minxuan Zhou, Chonghan Lee, Zheyu Li, Rishika Kushwah, Vijaykrishnan Narayanan, and Tajana Rosing. 2024. Primate: Processing in memory acceleration for dynamic token-pruning transformers. In *2024 29th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 557–563.

[218] Gyeonghwan Park, Sanghyeok Han, Byungkuk Yoon, and Jae-Joon Kim. 2025. DOTS: DRAM-PIM Optimization for Tall and Skinny GEMM Operations in LLM Inference. In *2025 Design, Automation & Test in Europe Conference (DATE)*. IEEE, 1–2.

[219] Jaehyun Park, Jaewan Choi, Kwanhee Kyung, Michael Jaemin Kim, Yongsuk Kwon, Nam Sung Kim, and Jung Ho Ahn. 2024. 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, Volume 2*. 103–119.

[220] Jaehyun Park, Byeongho Kim, Sungmin Yun, Eojin Lee, Minsoo Rhu, and Jung Ho Ahn. 2021. TRiM: Enhancing processor-memory interfaces with scalable tensor reduction in memory. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*. 268–281.

[221] Jaewoo Park, Sugil Lee, and Jongeun Lee. 2023. Ntt-pim: Row-centric architecture and mapping for efficient number-theoretic transform on pim. In *2023 60th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.

[222] Sang-Soo Park, KyungSoo Kim, Jinin So, Jin Jung, Jonggeon Lee, Kyoungwan Woo, Nayeon Kim, Younghyun Lee, Hyungyo Kim, Yongsuk Kwon, et al. 2024. An lpddr-based cxl-pnm platform for tco-efficient inference of transformer-based large language models. In *2024 IEEE International Symposium on High-Performance Computer Architecture*

(HPCA). IEEE, 970–982.

[223] Yongmo Park, Aporva Amarnath, Subhankar Pal, Karthik Swaminathan, Alper Buyuktosunoglu, Hayim Shaul, Ehud Aharoni, Nir Drucker, Wei D Lu, Omri Soceanu, et al. 2025. FHENDI: A Near-DRAM Accelerator for Compiler-Generated Fully Homomorphic Encryption Applications. In *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 1127–1142.

[224] Yongmo Park, Subhankar Pal, Aporva Amarnath, Karthik Swaminathan, Wei D Lu, Alper Buyuktosunoglu, and Pradip Bose. 2024. Dramaton: A near-dram accelerator for large number theoretic transforms. *IEEE Computer Architecture Letters* 23, 1 (2024), 108–111.

[225] Neel Patel, Amin Mamandipoor, Mohammad Nouri, and Mohammad Alian. 2024. SmartDIMM: in-memory acceleration of upper layer protocols. In *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 312–329.

[226] Neel Patel, Amin Mamandipoor, Derrick Quinn, and Mohammad Alian. 2023. Xfm: Accelerated software-defined far memory. In *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*. 769–783.

[227] 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)*. IEEE, 1275–1288.

[228] Siddhartha Raman Sundara Raman, Lizy John, and Jaydeep P Kulkarni. 2025. SPARK: Sparsity Aware, Low Area, Energy-Efficient, Near-memory Architecture for Accelerating Linear Programming Problems. In *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 99–112.

[229] Joseph Rogers, Taha Soliman, and Magnus Jahre. 2024. AIO: An abstraction for performance analysis across diverse accelerator architectures. In *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 487–500.

[230] Elaheh Sadredini, Reza Rahimi, Mohsen Imani, and Kevin Skadron. 2021. Sunder: Enabling low-overhead and scalable near-data pattern matching acceleration. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*. 311–323.

[231] Mohammadreza Saed, Prashant J Nair, and Tor M Aamodt. 2025. RayN: Ray Tracing Acceleration with Near-memory Computing. In *Proceedings of the 58th IEEE/ACM International Symposium on Microarchitecture®*. 277–291.

[232] SAITPublic. 2025. PIMSimulator: Processing-In-Memory (PIM) Simulator. https://github.com/SAITPublic/PIMSimulator. GitHub repository, accessed: 2025-11-06.

[233] Daniel Sanchez and Christos Kozyrakis. 2013. ZSim: Fast and accurate microarchitectural simulation of thousand-core systems. *ACM SIGARCH Computer architecture news* 41, 3 (2013), 475–486.

[234] Paulo C Santos, Bruno E Forlin, and Luigi Carro. 2021. Providing Plug N'Play for Processing-in-Memory Accelerators. In *Proceedings of the 26th Asia and South Pacific Design Automation Conference*. 651–656.

[235] Paulo C Santos, Bruno E Forlin, and Luigi Carro. 2021. Sim2pim: A fast method for simulating host independent & pim agnostic designs. In *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 226–231.

[236] Paulo C Santos, Geraldo F Oliveira, João P Lima, Marco AZ Alves, Luigi Carro, and Antonio CS Beck. 2018. Processing in 3D memories to speed up operations on complex data structures. In *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 897–900.

[237] Fabian Schuiki, Michael Schaffner, Frank K Gürkaynak, and Luca Benini. 2018. A scalable near-memory architecture for training deep neural networks on large in-memory datasets. *IEEE Trans. Comput.* 68, 4 (2018), 484–497.

[238] Brian C Schwedock and Nathan Beckmann. 2024. Leviathan: A Unified System for General-Purpose Near-Data Computing. In *2024 57th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 1278–1294.

[239] Minseok Seo, Xuan Truong Nguyen, Seok Joong Hwang, Yongkee Kwon, Guhyun Kim, Chanwook Park, Ilkon Kim, Jaehan Park, Jeongbin Kim, Woojae Shin, et al. 2024. Ianus: Integrated accelerator based on npu-pim unified memory system. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*. 545–560.

[240] Seong Hoon Seo, Junghoon Kim, Donghyun Lee, Seonah Yoo, Seokwon Moon, Yeonhong Park, and Jae W Lee. 2025. FACIL: Flexible DRAM Address Mapping for SoC-PIM Cooperative On-device LLM Inference. In *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 1720–1733.

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

[242] 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 *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*. 273–287.

- [243] Akhil Shekar, Kevin Gaffney, Martin Prammer, Khyati Kiyawat, Lingxi Wu, Helena Caminal, Zhenxing Fan, Yimin Gao, Ashish Venkat, José F Martínez, et al. 2025. Membrane: Accelerating Database Analytics with Bank-Level DRAM-PIM Filtering. *arXiv preprint arXiv:2504.06473* (2025).
- [244] Hyunsung Shin, Dongyoung Kim, Eunhyeok Park, Sungho Park, Yongsik Park, and Sungjoo Yoo. 2018. McDRAM: Low latency and energy-efficient matrix computations in DRAM. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 37, 11 (2018), 2613–2622.
- [245] Hoon Shin, Rihae Park, and Jae W Lee. 2024. A Processing-using-Memory Architecture for Commodity DRAM Devices with Enhanced Compatibility and Reliability. In *Proceedings of the 43rd IEEE/ACM International Conference on Computer-Aided Design*. 1–10.
- [246] Yongwon Shin, Juseong Park, Sungjun Cho, and Hyojin Sung. 2023. Pimflow: Compiler and runtime support for cnn models on processing-in-memory dram. In *Proceedings of the 21st ACM/IEEE International Symposium on Code Generation and Optimization*. 249–262.
- [247] Farzana Ahmed Siddique, Deyuan Guo, Zhenxing Fan, Mohammadhosein Gholamrezaei, Morteza Baradaran, Alif Ahmed, Hugo Abbot, Kyle Durrer, Kumaresh Nandagopal, Ethan Ermovick, et al. 2024. Architectural Modeling and Benchmarking for Digital DRAM PIM. In *2024 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 247–261.
- [248] Joonseop Sim, Soohong Ahn, Taeyoung Ahn, Seungyong Lee, Myunghyun Rhee, Jooyoung Kim, Kwangsik Shin, Donguk Moon, Euiseok Kim, and Kyoung Park. 2022. Computational cxl-memory solution for accelerating memory-intensive applications. *IEEE Computer Architecture Letters* 22, 1 (2022), 5–8.
- [249] Jaehyeong Sim, Hoseok Seol, and Lee-Sup Kim. 2018. NID: Processing binary convolutional neural network in commodity DRAM. In *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 1–8.
- [250] Gagandeep Singh, Juan Gómez-Luna, Giovanni Mariani, Geraldo F Oliveira, Stefano Corda, Sander Stuijk, Onur Mutlu, and Henk Corporaal. 2019. Napel: Near-memory computing application performance prediction via ensemble learning. In *Proceedings of the 56th annual design automation conference 2019*. 1–6.
- [251] Aman Sinha, Huei-Chun Yang, Pei-Yi Liu, Yen-Shi Kuo, Yuhao Fang, Tien-Shuo Chang, Ke-Han Li, and Bo-Cheng Lai. 2022. DSIM: Distributed sequence matching on near-dram accelerator for genome assembly. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 12, 2 (2022), 486–499.
- [252] Hyojun Son, Gilbert Jonatan, Xiangyu Wu, Haeyoon Cho, Kaustubh Shivdikar, José L Abellán, Ajay Joshi, David Kaeli, and John Kim. 2025. PIMnet: A Domain-Specific Network for Efficient Collective Communication in Scalable PIM. In *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 1557–1572.
- [253] JEDEC Standard. 2012. JEDEC standard. *Temperature cycling* (2012).
- [254] Chirag Sudarshan, Taha Soliman, Cecilia De la Parra, Christian Weis, Leonardo Ecco, Matthias Jung, Norbert Wehn, and Andre Guntoro. 2021. A novel DRAM-based process-in-memory architecture and its implementation for CNNs. In *Proceedings of the 26th Asia and South Pacific Design Automation Conference*. 35–42.
- [255] Weiyi Sun, Zhaoshi Li, Shouyi Yin, Shaojun Wei, and Leibo Liu. 2021. Abc-dimm: Alleviating the bottleneck of communication in dimm-based near-memory processing with inter-dimm broadcast. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 237–250.
- [256] Purab Ranjan Sutradhar, Sathwika Bavikadi, Mark Connolly, Savankumar Prajapati, Mark A Indovina, Sai Manoj Pudukotai Dinakarrao, and Amlan Ganguly. 2021. Look-up-table based processing-in-memory architecture with programmable precision-scaling for deep learning applications. *IEEE Transactions on Parallel and Distributed Systems* 33, 2 (2021), 263–275.
- [257] Purab Ranjan Sutradhar, Mark Connolly, Sathwika Bavikadi, Sai Manoj Pudukotai Dinakarrao, Mark A Indovina, and Amlan Ganguly. 2020. pPIM: A programmable processor-in-memory architecture with precision-scaling for deep learning. *IEEE Computer Architecture Letters* 19, 2 (2020), 118–121.
- [258] Nishil Talati, Haojie Ye, Yichen Yang, Leul Belayneh, Kuan-Yu Chen, David Blaauw, Trevor Mudge, and Ronald Dreslinski. 2022. Ndminer: accelerating graph pattern mining using near data processing. In *Proceedings of the 49th Annual International Symposium on Computer Architecture*. 146–159.
- [259] Hritvik Taneja and Moin Qureshi. 2025. DREAM: Enabling Low-Overhead Rowhammer Mitigation via Directed Refresh Management. In *Proceedings of the 52nd Annual International Symposium on Computer Architecture*. 776–792.
- [260] Yupeng Tang, Seung-seob Lee, Abhishek Bhattacharjee, and Anurag Khandelwal. 2025. pulse: Accelerating Distributed Pointer-Traversals on Disaggregated Memory. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*. 858–875.
- [261] Boyu Tian, Qihang Chen, and Mingyu Gao. 2023. Abndp: Co-optimizing data access and load balance in near-data processing. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*. 3–17.
- [262] Boyu Tian, Yiwei Li, Li Jiang, Shuangyu Cai, and Mingyu Gao. 2024. Ndpbridge: Enabling cross-bank coordination in near-dram-bank processing architectures. In *2024 ACM/IEEE 51st Annual International Symposium on Computer*

Architecture (ISCA). IEEE, 628–643.

[263] Diego G Tomé, Paulo C Santos, Luigi Carro, Eduardo C Almeida, and Marco AZ Alves. 2018. HIPE: HMC instruction predication extension applied on database processing. In *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 261–264.

[264] Joao Vieira, Nuno Roma, Gabriel Falcao, and Pedro Tomás. 2022. gem5-ndp: Near-data processing architecture simulation from low level caches to DRAM. In *2022 IEEE 34th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*. IEEE, 41–50.

[265] João Vieira, Nuno Roma, Gabriel Falcao, and Pedro Tomás. 2024. Ndpmulator: Enabling full-system simulation for near-data accelerators from caches to dram. *IEEE Access* 12 (2024), 10349–10365.

[266] Tobias Vinçon, Christian Knödler, Leonardo Solis-Vasquez, Arthur Bernhardt, Sajjad Tamimi, Lukas Weber, Florian Stock, Andreas Koch, and Ilia Petrov. 2022. Near-data processing in database systems on native computational storage under HTAP workloads. *Proceedings of the VLDB Endowment* 15, 10 (2022), 1991–2004.

[267] David Wang, Brinda Ganesh, Nuengwong Tuaycharoen, Kathleen Baynes, Aamer Jaleel, and Bruce Jacob. 2005. Dramsim: a memory system simulator. *ACM SIGARCH Computer Architecture News* 33, 4 (2005), 100–107.

[268] Junpeng Wang, Mengke Ge, Bo Ding, Qi Xu, Song Chen, and Yi Kang. 2023. Nicepim: Design space exploration for processing-in-memory dnn accelerators with 3-d stacked-dram. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 43, 5 (2023), 1456–1469.

[269] Yi Wang, Weixuan Chen, Jing Yang, and Tao Li. 2018. Towards memory-efficient allocation of CNNs on processing-in-memory architecture. *IEEE Transactions on Parallel and Distributed Systems* 29, 6 (2018), 1428–1441.

[270] Yitu Wang, Shiyu Li, Qilin Zheng, Linghao Song, Zongwang Li, Andrew Chang, Yiran Chen, et al. 2024. NDSEARCH: Accelerating graph-traversal-based approximate nearest neighbor search through near data processing. In *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 368–381.

[271] Zhengrong Wang, Christopher Liu, Aman Arora, Lizy John, and Tony Nowatzki. 2023. Infinity stream: Portable and programmer-friendly in-/near-memory fusion. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*. 359–375.

[272] Zhengrong Wang, Christopher Liu, Nathan Beckmann, and Tony Nowatzki. 2023. Affinity alloc: Taming not-so near-data computing. In *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*. 784–799.

[273] Zhengrong Wang, Jian Weng, Sihao Liu, and Tony Nowatzki. 2022. Near-stream computing: General and transparent near-cache acceleration. In *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 331–345.

[274] Tai-Hao Wen, Je-Min Hung, Hung-Hsi Hsu, Yuan Wu, Fu-Chun Chang, Chung-Yuan Li, Chih-Han Chien, Chin-I Su, Win-San Khwa, Jui-Jen Wu, et al. 2023. A 28nm nonvolatile AI edge processor using 4Mb analog-based near-memory-compute ReRAM with 27.2 TOPS/W for tiny AI edge devices. In *2023 IEEE Symposium on VLSI Technology and Circuits (VLSI Technology and Circuits)*. IEEE, 1–2.

[275] Lingxi Wu, Rasool Sharifi, Marzieh Lenjani, Kevin Skadron, and Ashish Venkat. 2021. Sieve: Scalable in-situ DRAM-based accelerator designs for massively parallel k-mer matching. In *2021 ACM/IEEE 48th annual international symposium on computer architecture (ISCA)*. IEEE, 251–264.

[276] Puqing Wu, Minhui Xie, Enrui Zhao, Dafang Zhang, Jing Wang, Xiao Liang, Kai Ren, and Yunpeng Chai. 2025. Turbocharge {ANNS} on Real {Processing-in-Memory} by Enabling {Fine-Grained}{Per-PIM-Core} Scheduling. In *2025 USENIX Annual Technical Conference (USENIX ATC 25)*. 1223–1241.

[277] Yuting Wu, Ziyu Wang, and Wei D Lu. 2024. Pim gpt a hybrid process in memory accelerator for autoregressive transformers. *npj Unconventional Computing* 1, 1 (2024), 4.

[278] Yao Xiao, Shahin Nazarian, and Paul Bogdan. 2018. Prometheus: Processing-in-memory heterogeneous architecture design from a multi-layer network theoretic strategy. In *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 1387–1392.

[279] Shanshan Xie, Can Ni, Aseem Sayal, Pulkit Jain, Fatih Hamzaoglu, and Jaydeep P Kulkarni. 2021. 16.2 eDRAM-CIM: Compute-in-memory design with reconfigurable embedded-dynamic-memory array realizing adaptive data converters and charge-domain computing. In *2021 IEEE International Solid-State Circuits Conference (ISSCC)*, Vol. 64. IEEE, 248–250.

[280] Tongxin Xie, Zhenhua Zhu, Bing Li, Yukai He, Cong Li, Guangyu Sun, Huazhong Yang, Yuan Xie, and Yu Wang. 2025. UniNDP: A Unified Compilation and Simulation Tool for Near DRAM Processing Architectures. In *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 624–640.

[281] Xinfeng Xie, Peng Gu, Jiayi Huang, Yufei Ding, and Yuan Xie. 2021. MPU-Sim: A simulator for in-DRAM near-bank processing architectures. *IEEE Computer Architecture Letters* 21, 1 (2021), 1–4.

[282] Xinfeng Xie, Zheng Liang, Peng Gu, Abanti Basak, Lei Deng, Ling Liang, Xing Hu, and Yuan Xie. 2021. SpaceA: Sparse matrix vector multiplication on processing-in-memory accelerator. In *2021 IEEE International Symposium on*

High-Performance Computer Architecture (HPCA). IEEE, 570–583.

[283] Xin Xin, Youtao Zhang, and Jun Yang. 2019. Roc: Dram-based processing with reduced operation cycles. In *Proceedings of the 56th Annual Design Automation Conference 2019*. 1–6.

[284] 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)*. IEEE, 303–314.

[285] Sheng Xu, Xiaoming Chen, Ying Wang, Yinhe Han, and Xiaowei Li. 2019. CuckooPIM: An efficient and less-blocking coherence mechanism for processing-in-memory systems. In *Proceedings of the 24th Asia and South Pacific Design Automation Conference*. 140–145.

[286] Sheng Xu, Xiaoming Chen, Ying Wang, Yinhe Han, Xuehai Qian, and Xiaowei Li. 2018. PIMSim: A flexible and detailed processing-in-memory simulator. *IEEE Computer Architecture Letters* 18, 1 (2018), 6–9.

[287] Sheng Xu, Ying Wang, Yinhe Han, and Xiaowei Li. 2018. PIMCH: Cooperative memory prefetching in processing-in-memory architecture. In *2018 23rd Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 209–214.

[288] A Giray Yağlikçi, Minesh Patel, Jeremie S Kim, Roknoddin Azizi, Ataberk Olgun, Lois Orosa, Hasan Hassan, Jisung Park, Konstantinos Kanellopoulos, Taha Shahroodi, et al. 2021. Blockhammer: Preventing rowhammer at low cost by blacklisting rapidly-accessed dram rows. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 345–358.

[289] Tao Yang, Hui Ma, Yilong Zhao, Fangxin Liu, Zhezhi He, Xiaoli Sun, and Li Jiang. 2023. Pimpr: Pim-based personalized recommendation with heterogeneous memory hierarchy. In *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 1–6.

[290] Weidong Yang, Yuqing Yang, Shuya Ji, Jianfei Jiang, Naifeng Jing, Qin Wang, Zhigang Mao, and Weiguang Sheng. 2024. Recpim: Efficient in-memory processing for personalized recommendation inference using near-bank architecture. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 43, 10 (2024), 2854–2867.

[291] Yuqing Yang, Weidong Yang, Qin Wang, Naifeng Jing, Jianfei Jiang, Zhigang Mao, and Weiguang Sheng. 2023. An efficient near-bank processing architecture for personalized recommendation system. In *Proceedings of the 28th Asia and South Pacific Design Automation Conference*. 122–127.

[292] Chao Yu, Sihang Liu, and Samira Khan. 2021. Multipim: A detailed and configurable multi-stack processing-in-memory simulator. *IEEE Computer Architecture Letters* 20, 1 (2021), 54–57.

[293] Seunghyuk Yu, Hyeonu Kim, Kyoungho Jeun, Sunyoung Hwang, Seongmin Cho, and Eojin Lee. 2025. ComPASS: A Compatible PIM Protocol Architecture and Scheduling Solution for Processor-PIM Collaboration. In *Proceedings of the 58th IEEE/ACM International Symposium on Microarchitecture®*. 49–62.

[294] Zhibin Yu, Zhendong Bei, and Xuehai Qian. 2018. Datasize-aware high dimensional configurations auto-tuning of in-memory cluster computing. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*. 564–577.

[295] Zhiheng Yue, Huizheng Wang, Jiahao Fang, Jinyi Deng, Guangyang Lu, Fengbin Tu, Ruiqi Guo, Yuxuan Li, Yubin Qin, Yang Wang, et al. 2024. Exploiting similarity opportunities of emerging vision ai models on hybrid bonding architecture. In *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 396–409.

[296] Zhiheng Yue, Yang Wang, Chao Li, Shaojun Wei, Yang Hu, and Shouyi Yin. 2025. 3D-PATH: A Hierarchy LUT Processing-in-memory Accelerator with Thermal-aware Hybrid Bonding Integration. In *Proceedings of the 58th IEEE/ACM International Symposium on Microarchitecture®*. 78–93.

[297] Ismail Emir Yuksel, Akash Sood, Ataberk Olgun, Oğuzhan Canpolat, Haocong Luo, Nisa Bostanci, Mohammad Sadrosadati, Giray Yaglikci, and Onur Mutlu. 2025. PuDHammer: Experimental Analysis of Read Disturbance Effects of Processing-using-DRAM in Real DRAM Chips. In *Proceedings of the 52nd Annual International Symposium on Computer Architecture*. 757–775.

[298] İsmail Emir Yüksel, Yahya Can Tuğrul, Ataberk Olgun, F Nisa Bostancı, A Giray Yağlıkçı, Geraldo F Oliveira, Haocong Luo, Juan Gómez-Luna, Mohammad Sadrosadati, and Onur Mutlu. 2024. Functionally-complete boolean logic in real dram chips: Experimental characterization and analysis. In *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 280–296.

[299] Sungmin Yun, Byeongho Kim, Jaehyun Park, Hwayong Nam, Jung Ho Ahn, and Eojin Lee. 2022. GraNDe: Near-data processing architecture with adaptive matrix mapping for graph convolutional networks. *IEEE Computer Architecture Letters* 21, 2 (2022), 45–48.

[300] Sungmin Yun, Kwanhee Kyung, Juhwan Cho, Jaewan Choi, Jongmin Kim, Byeongho Kim, Sukhan Lee, Kyomin Sohn, and Jung Ho Ahn. 2024. Duplex: A device for large language models with mixture of experts, grouped query attention, and continuous batching. In *2024 57th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 1429–1443.

[301] Sungmin Yun, Hwayong Nam, Kwanhee Kyung, Jaehyun Park, Byeongho Kim, Yongsuk Kwon, Eojin Lee, and Jung Ho Ahn. 2024. CLAY: CXL-based Scalable NDP Architecture Accelerating Embedding Layers. In *Proceedings of the 38th ACM International Conference on Supercomputing*. 338–351.

[302] Sungmin Yun, Hwayong Nam, Jaehyun Park, Byeongho Kim, Jung Ho Ahn, and Eojin Lee. 2023. GraNDe: Efficient near-data processing architecture for graph neural networks. *IEEE Trans. Comput.* 73, 10 (2023), 2391–2404.

[303] Chao Zhang, Tong Meng, and Guangyu Sun. 2018. Pm3: Power modeling and power management for processing-in-memory. In *2018 IEEE International symposium on high performance computer architecture (HPCA)*. IEEE, 558–570.

[304] Dongping Zhang, Nuwan Jayasena, Alexander Lyashevsky, Joseph L Greathouse, Lifan Xu, and Michael Ignatowski. 2014. TOP-PIM: Throughput-oriented programmable processing in memory. In *Proceedings of the 23rd international symposium on High-performance parallel and distributed computing*. 85–98.

[305] Fan Zhang, Shaahin Angizi, and Deliang Fan. 2021. Max-PIM: Fast and efficient max/min searching in DRAM. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 211–216.

[306] Jian Zhang, Yujie Ren, Marie Nguyen, Changwoo Min, and Sudarsun Kannan. 2024. {OmniCache}: Collaborative Caching for Near-storage Accelerators. In *22nd USENIX Conference on File and Storage Technologies (FAST 24)*. 35–50.

[307] Mingxing Zhang, Youwei Zhuo, Chao Wang, Mingyu Gao, Yongwei Wu, Kang Chen, Christos Kozyrakis, and Xuehai Qian. 2018. GraphP: Reducing communication for PIM-based graph processing with efficient data partition. In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 544–557.

[308] Xueyong Zhang and Arindam Basu. 2022. A 915–1220 TOPS/W, 976–1301 GOPS hybrid in-memory computing based always-on image processing for neuromorphic vision sensors. *IEEE Journal of Solid-State Circuits* 58, 3 (2022), 589–599.

[309] Xingyao Zhang, Shuaiwen Leon Song, Chenhao Xie, Jing Wang, Weigong Zhang, and Xin Fu. 2020. Enabling highly efficient capsule networks processing through a PIM-based architecture design. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 542–555.

[310] Yilong Zhao, Mingyu Gao, Fangxin Liu, Yiwei Hu, Zongwu Wang, Han Lin, Ji Li, He Xian, Hanlin Dong, Tao Yang, et al. 2024. Um-pim: Dram-based pim with uniform & shared memory space. In *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 644–659.

[311] Yang Katie Zhao, Shang Wu, Jingqun Zhang, Sixu Li, Chaojian Li, and Yingyan Celine Lin. 2023. Instant-nerf: Instant on-device neural radiance field training via algorithm-accelerator co-designed near-memory processing. In *2023 60th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.

[312] Jiapei Zheng, Hao Jiang, Xinkai Nie, Zhangcheng Huang, Chixiao Chen, and Qi Liu. 2023. Tipu: A spatial-locality-aware near-memory tile processing unit for 3d point cloud neural network. In *2023 60th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.

[313] Minxuan Zhou, Yunhui Guo, Weihong Xu, Bin Li, Kevin W Eliceiri, and Tajana Rosing. 2021. MAT: Processing in-memory acceleration for long-sequence attention. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 25–30.

[314] Minxuan Zhou, Mohsen Imani, Saransh Gupta, and Tajana Rosing. 2019. Thermal-aware design and management for search-based in-memory acceleration. In *Proceedings of the 56th Annual Design Automation Conference 2019*. 1–6.

[315] Minxuan Zhou, Mohsen Imani, Yeseong Kim, Saransh Gupta, and Tajana Rosing. 2021. Dp-sim: A full-stack simulation infrastructure for digital processing in-memory architectures. In *Proceedings of the 26th Asia and South Pacific Design Automation Conference*. 639–644.

[316] Minxuan Zhou, Muzhou Li, Mohsen Imani, and Tajana Rosing. 2021. Hygraph: Accelerating graph processing with hybrid memory-centric computing. In *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 330–335.

[317] Minxuan Zhou, Xuan Wang, and Tajana Rosing. 2023. OverlaPIM: Overlap optimization for processing in-memory neural network acceleration. In *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 1–6.

[318] Minxuan Zhou, Weihong Xu, Jaeyoung Kang, and Tajana Rosing. 2022. Transpim: A memory-based acceleration via software-hardware co-design for transformer. In *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 1071–1085.

[319] Ranyang Zhou, Arman Roohi, Durga Misra, and Shaahin Angizi. 2022. ReD-LUT: Reconfigurable in-DRAM LUTs enabling massive parallel computation. In *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*. 1–8.

[320] Ranyang Zhou, Sepehr Tabrizchi, Mehrdad Morsali, Arman Roohi, and Shaahin Angizi. 2023. P-pim: A parallel processing-in-dram framework enabling row hammer protection. In *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 1–6.

[321] Zhe Zhou, Cong Li, Xuechao Wei, Xiaoyang Wang, and Guangyu Sun. 2022. GNNear: Accelerating full-batch training of graph neural networks with near-memory processing. In *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques*. 54–68.

[322] Zhe Zhou, Cong Li, Fan Yang, and Guangyu Sun. 2023. Dimm-link: Enabling efficient inter-dimm communication for near-memory processing. In *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 302–316.

[323] Haozhe Zhu, Bo Jiao, Jinshan Zhang, Xinru Jia, Yunzhengmao Wang, Tianchan Guan, Shengcheng Wang, Dimin Niu, Hongzhong Zheng, Chixiao Chen, et al. 2022. COMB-MCM: Computing-on-memory-boundary NN processor with bipolar bitwise sparsity optimization for scalable multi-chiplet-module edge machine learning. In *2022 IEEE International Solid-State Circuits Conference (ISSCC)*, Vol. 65. IEEE, 1–3.

[324] Zhenhua Zhu, Jun Liu, Guohao Dai, Shulin Zeng, Bing Li, Huazhong Yang, and Yu Wang. 2023. Processing-in-hierarchical-memory architecture for billion-scale approximate nearest neighbor search. In *2023 60th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.

[325] Youwei Zhuo, Chao Wang, Mingxing Zhang, Rui Wang, Dimin Niu, Yanzhi Wang, and Xuehai Qian. 2019. Graphq: Scalable pim-based graph processing. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*. 712–725.

[326] Farzaneh Zokaee, Fan Chen, Guangyu Sun, and Lei Jiang. 2022. Sky-sorter: A processing-in-memory architecture for large-scale sorting. *IEEE Trans. Comput.* 72, 2 (2022), 480–493.