computers

MDPI

Systematic Review

Redefining Trust at the Memory Wall: The Threat Landscape, an Adversary Model and a Conceptual Framework for Secure Processing-in-Memory

Sayali Waingankar [1,*], Hung Q. Le [1] and Driss Benhaddou [2,*]

[1] Electrical and Computer Engineering Department, Cullen College of Engineering, University of Houston, Houston, TX 77004, USA; hql50577@central.uh.edu
[2] Electrical Engineering Department, College of Engineering, Alfaisal University, Riyadh 11533, Saudi Arabia
* Correspondence: sswaingankar@uh.edu (S.W.); dbenhaddou@alfaisal.edu (D.B.)

Abstract

Processing-in-Memory (PIM) architectures are transforming system hierarchies by embedding computation within memory and collapsing traditional CPU–memory separation. This co-location enables measured gains in throughput and energy efficiency for data-intensive workloads such as AI inference and graph analytics, but executing untrusted logic inside memory introduces attack surfaces that CPU-centric security models are not designed to address. Side-channel leakage, privilege escalation within PIM logic units, and tenant isolation violations are structurally enabled by the same physical proximity which makes PIM efficient. This survey examines these threats systematically, drawing on 47 systematically selected publications and supplementary references from architecture, systems security, and hardware design venues covering PIM-specific literature from 2015 onward. We developed a structured threat model that characterizes assets, attacker capabilities, and trust boundaries at subarrays and interconnect granularity for both near-bank and near-memory PIM classes. We survey and classify existing attack vectors and countermeasures, organizing them according to the architectural properties they exploit or on which they depend. Drawing on gaps identified in the surveyed literature, we propose a conceptual secure-by-design framework that locates trust enforcement within the memory substrate itself, using physical data locality, internal bandwidth constraints, and memory-disaggregation topology as enforcement primitives rather than adapting CPU-centric enclave models. This framework is a structured design proposal, not a demonstrated implementation. The threat model, defense taxonomy, and design framework constitute a reference baseline for researchers building secure PIM systems and a gap map for future experimental and formal verification.

Keywords: data remanence; hardware security; near-data processing; processing-in-memory; RowHammer; side-channel attacks; trusted execution environment

Academic Editors: Paolo Bellavista, Ron (Rongyu) Lin and Taskin Kocak

Received: 16 June 2026
Revised: 8 September 2026
Accepted: 8 September 2026
Published: 11 September 2026

1. Introduction

The exponential growth of data-intensive workloads, driven by large-scale machine learning inference, graph analytics, and encrypted database processing, has exposed a fundamental limitation in conventional von Neumann architectures: the physical separation of memory and compute creates a data movement bottleneck that degrades both performance and energy efficiency [1,2]. Empirical measurements on mobile consumer workloads confirm that data-intensive pipelines spend more than 60% of the execution time and up to

70% of the system energy on memory access rather than arithmetic [3]. Similar bottlenecks have been observed in server-scale graph analytics and ML inference [4,5]. As memory bandwidth scales more slowly than compute throughput, these bottlenecks compound with each processor generation, and workloads requiring frequent access to large, sensitive datasets amplify it further.

Processing-in-memory (PIM) directly addresses this bottleneck by integrating computational logic within or adjacent to memory arrays, thereby reducing the energy and latency cost of data movement [4,5]. Commercial DRAM-PIM modules report 2–10× lower energy per operation and 20–93× speedups over CPU and GPU baselines in graph mining, streaming analytics, and AI inference [6,7]. Architectures targeting a large language model (LLM) inference achieve up to 6.94× lower total cost of ownership per query-per-second and 10–20× less energy per token relative to state-of-the-art GPUs [8]. These performance characteristics make PIM an active deployment target across cloud, mobile, and edge contexts [9,10], each of which introduces distinct security requirements that the architecture was not originally designed to satisfy.

PIM's structural departure from conventional processor–DRAM interfaces, specifically the placement of compute logic directly alongside unencrypted memory contents without the isolation guarantees of a processor memory-protection unit, creates an attack surface with no direct precedent in prior memory subsystem designs. RowHammer attacks on DRAM-PIM arrays exploit activation patterns generated by near-memory compute units; Blacksmith-class non-uniform hammering sequences bypass TRR on all tested DDR4 DIMMs [11], and PIM's internally generated workload-dependent row activations are structurally indistinguishable from adversarial hammering sequences from the host refresh controller's perspective [12,13]. Emerging non-volatile memory (NVM) technologies such as ReRAM and PCM introduce persistent fault susceptibility and endurance-related degradation that compounds across banks. Carefully crafted wear-out patterns can embed latent faults across crossbar arrays, and resistance-state corruption from repeated adversarial writes degrades read margins over time in ways that standard error-correction cannot fully mask [14,15]. In multi-tenant cloud deployments, where multiple workloads share PIM hardware, these physical-layer vulnerabilities compose with software-layer isolation failures to produce cross-layer exploits for which no complete mitigation currently exists.

Prior work has addressed subsets of this problem. Mutlu and Kim [16] provided a detailed retrospective analysis of DRAM disturbance attacks and early isolation techniques but restricted their scope to DRAM-based RowHammer vulnerabilities, which do not address cross-layer threats, Near-Data Processing (NDP) architectures or NVM technologies. Arafin and Lu [17] identified architectural security challenges in PIM systems but offered no systematic review methodology, quantitative performance–security budget analysis, or formal adversary model. The convergence of NDP, Logic-in-Memory (LiM), and NVM-based PIM in production deployments, combined with the emergence of multi-tenant workloads and firmware-accessible near-bank compute units, creates a gap that neither work addresses. This survey addresses this gap through a systematic synthesis spanning all three PIM architecture classes. Specifically, the gap spans four dimensions: (i) no unified adversary model consolidates attacker capabilities across host firmware, interconnect protocols, and subarray-level fault phenomena; (ii) no systematic threat classification organizes attack vectors and countermeasures across LiM, NDP, and NVM-based PIM under a common analytical structure; (iii) reported defense overheads have not been cross-compared quantitatively across heterogeneous evaluation platforms; and (iv) no composable security framework addresses the interaction effects between defenses operating at adjacent architectural layers.

To address these dimensions, this survey synthesizes the PIM security literature into a unified analytical structure and identifies specific design requirements for a memory-native security framework. The following list summarizes the main contributions.

The main contributions of this paper are as follows:

1. A PRISMA-aligned systematic review of 47 publications on PIM security, spanning Logic-in-Memory (LiM), Near-Data Processing (NDP), and Non-Volatile Memory (NVM)-based architectures.
2. A six-domain threat classification mapping 13 representative defense mechanisms across the surveyed corpus, with quantitative synthesis of reported runtime and area overheads across each defense category.
3. A structured adversary model for PIM systems that identifies four classes of attackers with explicit capability boundaries, excluded capabilities, and success conditions. The model applies a STRIDE-inspired methodology adapted to the memory-compute threat surface.
4. A conceptual secure-by-design framework that integrates five composable memory-native security primitives: subarray-level isolation, temporal data lifecycle management (secure scrubbing and zeroization), distributed trust anchors, data-oblivious execution, and adaptive policy orchestration. The framework is presented as a structured design proposal supported by preliminary DRAM-interface sensitivity analysis, not as a demonstrated hardware implementation. It consolidates design requirements emerging from the gap analysis and motivates subsequent full-system experimental validation.
5. Identification of four open research trajectories, threat-aware design, standardized attestation interfaces, AI-aware multi-tenant models, and ecosystem coordination, each grounded in gaps identified in Sections 5 and 6.

The remainder of this paper is organized as follows. Section 2 describes the review methodology. Section 3 provides background on PIM architectures. Section 4 characterizes the attack surface. Section 5 presents threat classification and defense analysis. Section 6 formalizes the adversary model. Section 7 introduces the secure-by-design conceptual framework. Section 8 identifies open research trajectories and concludes the paper.

2. Review Methodology

This review was conducted and is reported in accordance with the PRISMA 2020 statement [18]. A completed PRISMA 2020 checklist for this review is provided in the Supplementary Materials. Our objective was to systematically identify, screen, and synthesize research on security threats and mitigation in Processing-in-Memory (PIM) architectures.

Searches were executed in April 2025 across five databases: IEEE Xplore, ACM Digital Library, SpringerLink, ScienceDirect, and arXiv (categories cs.AR and cs.CR). Three Boolean query strings targeted distinct threat domains. Query 1 combined PIM architectural terms with security keywords: (“processing-in-memory” OR “near-data processing” OR “compute-in-memory” OR “logic-in-memory” OR “PIM” OR “NDP”) AND (“security” OR “privacy” OR “trusted execution” OR “isolation” OR “side-channel” OR “RowHammer” OR “covert channel” OR “data remanence” OR “threat model” OR “attestation” OR “enclave” OR “hardware security”). Query 2 targeted NVM-specific security mechanisms: (“non-volatile memory” OR “persistent memory” OR “ReRAM” OR “PCM” OR “NVM”) AND (“security” OR “remanence” OR “integrity” OR “side-channel” OR “encryption” OR “rollback” OR “erasure”) AND (“processing-in-memory” OR “near-data processing” OR “PIM” OR “in-memory”). Query 3 targeted DRAM disturbance attacks in PIM contexts: (“RowHammer” OR “RowPress” OR “DRAM disturbance” OR “bit flip” OR “TRR” OR

"target row refresh") AND ("processing-in-memory" OR "near-data processing" OR "PIM" OR "in-DRAM" OR "memory security" OR "isolation").

All searches applied a date filter of January 2014 to April 2025 and were restricted to peer-reviewed journals, conference and symposium proceedings, and arXiv preprints of at least eight pages with a concrete implementation or formal model. Language was restricted to English.

A record was included if all of the following held: (IC1) the paper addresses PIM, NDP, CiM, LiM, or NVM-PIM architecture explicitly; (IC2) the paper contains a defined threat model, demonstrated attack, or evaluated defense mechanism with PIM-specific scope; (IC3) publication date falls within 2014–2025; (IC4) the venue is peer-reviewed or the arXiv preprint meets the depth criterion above; (IC5) full text is available in English.

A record was excluded if any of the following held: (EC1) security content is absent, addressing only performance or area; (EC2) the work addresses general trusted-execution or enclave security without PIM-specific threat analysis; (EC3) the work is a white paper or vendor note without peer review; (EC4) full text was unavailable; (EC5) security analysis addresses only standard DRAM reliability such as ECC without PIM-specific threat analysis.

Initial retrieval yielded 930 records from IEEE Xplore, 2857 from ACM Digital Library, 2170 from SpringerLink, 57 from ScienceDirect, and 61 from arXiv, totaling 6075 records. ACM Digital Library exports were retrieved in batches due to platform constraints; 91 within-platform duplicates were identified and removed during consolidation. Eight additional records were identified through backward reference chaining of retained works. After cross-database deduplication by DOI and normalized title string, 4838 unique records remained for screening.

Title-and-abstract screening was conducted manually by the primary author. To mitigate single-reviewer bias, inclusion and exclusion criteria were iteratively refined through discussion among all co-authors before screening commenced, and the final synthesis, including threat classification assignments and defense-category mappings, was reviewed and validated by all co-authors. All screening and extraction decisions are transparently reported against the stated criteria (IC1–IC5, EC1–EC5) to enable independent verification, and a completed PRISMA 2020 checklist is provided in the Supplementary Materials. Applying criteria IC1–IC5 and EC1–EC5 excluded 4629 records; 209 proceeded to full-text review alongside the 8 reference-chained records, for a total of 217 full-text assessments. Full-text screening excluded 170 records: 101 addressed PIM performance or area without security content (EC1); 45 addressed general trusted-execution or enclave security without PIM-specific threat analysis (EC2); 19 were white papers or lacked sufficient implementation detail (EC3); and 5 identified through reference chaining did not meet inclusion criterion IC2. Forty-four publications were retained from database searches and 3 from reference chaining, NeuroPIM, Virtual PIM, and SecPM, yielding a final synthesis corpus of 47 publications. Figure 1 illustrates the complete PRISMA flow.

For each retained publication, the following fields were extracted into a structured coding table: (i) PIM architecture class (LiM, NDP, or NVM-based); (ii) threat category addressed (using the six-domain classification developed in Section 5); (iii) adversary model assumptions (attacker capability level, trust boundaries, and scope exclusions); (iv) defense mechanism type and granularity; (v) evaluation platform (simulation, FPGA, or real hardware) and workload suite; and (vi) reported quantitative metrics, including performance overhead, area overhead, energy overhead, and security metrics such as covert-channel throughput or bit-flip rates, where available. Synthesis was conducted through thematic analysis: extracted data were iteratively grouped by threat domain and architectural boundary, producing the six-domain threat classification and the comparative defense summary in Section 5.7. Cross-paper overhead comparison is also presented with

explicit caveats regarding heterogeneous evaluation baselines. No meta-analytic pooling was performed, as the heterogeneity of evaluation platforms, workload suites, and reported metrics across the corpus precludes meaningful statistical aggregation.

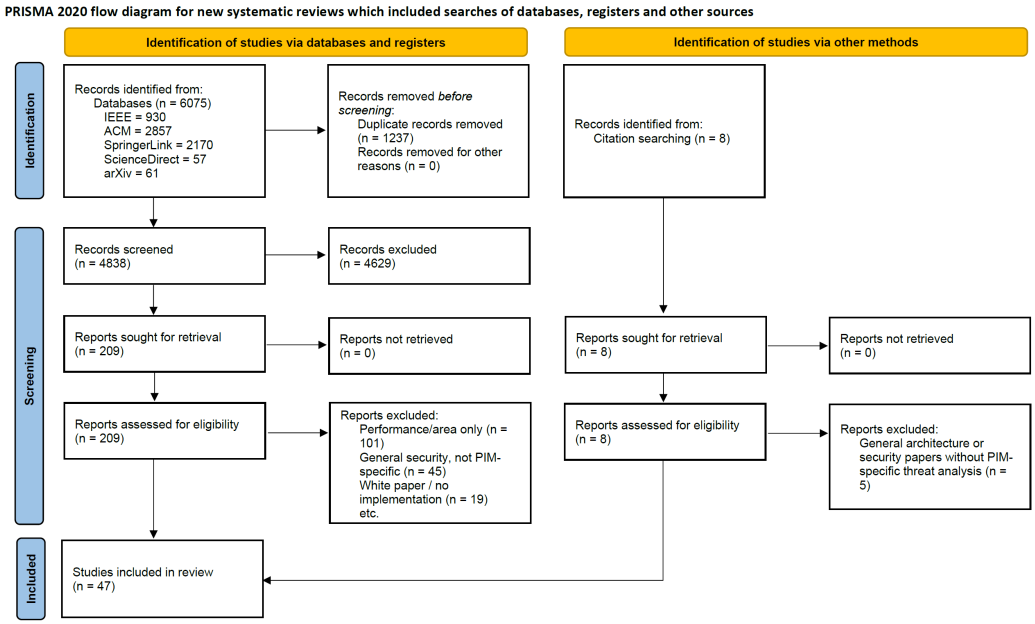


Figure 1. PRISMA 2020 flow diagram. Searches executed April-2025 across five databases using three Boolean query strings. Three papers identified through reference chaining (NeuroPIM, Virtual-PIM, SecPM) were assessed separately from database results. EC1–EC3 denote full-text exclusion criteria defined in Section 2. Diagram structure adapted from the PRISMA 2020 flow diagram template [18].

This survey does not cover the following: (i) general-purpose DRAM reliability or ECC mechanisms without PIM-specific threat analysis; (ii) software-only encryption or privacy-preserving computation frameworks (e.g., homomorphic encryption) that do not interact with PIM hardware; (iii) supply-chain attacks requiring chip decapsulation or invasive physical modification, which are noted as an open gap but excluded from the adversary model; (iv) security analysis of analog compute-in-memory designs operating purely through charge-sharing without any digital controller, as no published PIM-specific security mechanism targets this substrate; and (v) commercial PIM products released after the search cutoff date of April 2025. The conceptual framework proposed in Section 7 is a structured design proposal supported by preliminary DRAM-interface sensitivity analysis, not a demonstrated hardware implementation; its validation requirements are specified in Section 7 and scoped as future work.

3. Types of Processing-in-Memory (PIM) Architectures

The attack surface of Processing-in-Memory (PIM) is diverse, encompassing designs that embed simple logic directly within memory arrays as well as architectures that integrate programmable cores near memory banks. This diversity is architectural and functional, spanning differences in programmability, workload scope, and underlying memory technologies. As illustrated in Figure 2, compute placement fundamentally shapes these design classes, Logic-in-Memory (LiM), Near-Data Processing (NDP), and Non-Volatile Memory (NVM)-based PIM.

Logic-in-Memory (LiM) designs embed computation directly inside memory arrays, typically exploiting the analog properties of DRAM cells [19–21]. Ambit, a seminal DRAM-based PIM design, employs triple-row activation to realize bulk bitwise operations such as AND, OR, and XOR through charge-sharing mechanisms [22]. This reduces data movement

and can accelerate bulk bitwise database query workloads by up to 30× throughput over CPU baselines [23]. However, embedding logic at the row-buffer level creates security vulnerabilities specific to row-buffer logic. For example, predictable access patterns in Ambit's bulk operations can amplify disturbance errors, exacerbating RowHammer-style attacks [24]. Separately, since LiM relies on non-standard row activation sequences, adversaries can potentially exploit timing anomalies as side-channels, confirming the need for refresh-based mitigation or row remapping [25].

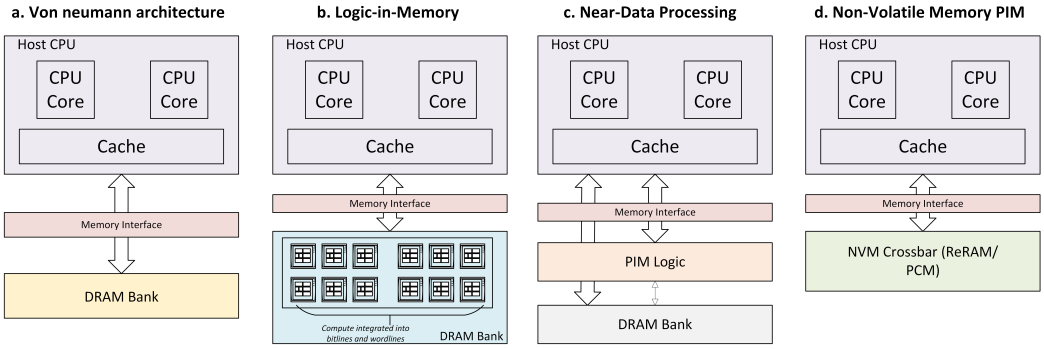


Figure 2. Types of Processing-in-Memory Architectures. (**a**) Von Neumann architecture, where all computation occurs in the host CPU and data must traverse the memory interface (shaded pink) to reach the DRAM bank (shaded yellow). (**b**) Logic-in-Memory (LiM), where compute logic is integrated directly into the DRAM bitlines and wordlines (shaded blue). (**c**) Near-Data Processing (NDP), where a discrete PIM logic layer (shaded orange) is placed adjacent to the DRAM bank. (**d**) Non-Volatile Memory PIM, where computation is performed within NVM crossbar arrays such as ReRAM or PCM (shaded green).

Near-Data Processing (NDP) architectures move computation into programmable logic placed close to, but outside, the memory array. Tesseract integrates simple in-order cores beneath stacked DRAM layers in an HBM package to accelerate graph workloads, reporting up to 10× performance improvements over CPU baselines [26]. Similarly, UPMEM's commercial PIM-DRAM DIMMs deploy thousands of embedded RISC cores adjacent to DRAM banks, enabling data-parallel workloads such as genomics and search [10]. While offering programmability and broader workload coverage than LiM, NDP introduces security challenges rooted in its physical organization. The proximity of programmable logic to shared memory arrays expands the attack surface for direct memory access (DMA) exploits, privileged software attacks, and multi-tenant isolation failures. In UPMEM, the absence of built-in memory encryption means security must be enforced externally, raising concerns for cloud-scale adoption. Thermal coupling between active cores and DRAM also complicates integrity guarantees, as overheating can induce retention faults exploitable by adversaries [27].

Beyond DRAM-based designs, ReRAM, PCM, and STT-RAM enable analog computation within dense crossbar arrays [28–33]. PRIME leverages ReRAM's resistive states to implement matrix–vector multiplication (MVM), a critical primitive for deep learning, achieving up to 895× energy efficiency gains over neural processing units [14]; tools such as NeuroSim extend this to full-stack simulation across multi-layer networks [34]. However, the shift from digital to analog in-memory computation introduces previously uncharacterized information leakage mechanisms. Device variability and resistance drift can leak information about model weights or input data, forming a new class of analog side-channels. Limited endurance also raises concerns, as carefully crafted wear-out patterns could embed persistent backdoors into the crossbar array. Security measures such as device-level error correction or stochastic bit masking mitigate reliability faults but add performance overhead, reflecting the trade-offs between efficiency and resilience [15].

A growing class of PIM architectures combines programmable digital logic with analog compute elements, creating hybrid designs that do not fit cleanly into the LiM, NDP, or NVM-only categories above. For example, architectures that pair ReRAM crossbar arrays for analog matrix–vector multiplication with digital RISC-V controllers for non-linear activation functions and data routing [29] inherit security exposures from both domains: the analog compute path is susceptible to resistance-drift side channels and device-variability leakage characteristic of NVM-based PIM, while the digital controller path faces code-injection and privilege-escalation risks characteristic of NDP. The security profile of such mixed-signal designs differs from pure digital systems in two specific ways. First, the analog-to-digital conversion boundary introduces a new information leakage surface, as ADC quantization noise can be correlated with input data to reconstruct operand values. Second, the absence of a unified memory protection model spanning both analog crossbar state and digital scratchpad memory means that isolation enforcement must operate across two physically distinct substrates with incompatible access semantics. These hybrid designs are not separately classified in our taxonomy because no PIM-specific security mechanism in the surveyed literature targets the analog–digital boundary as a distinct threat surface; we identify this as an open gap.

The variety of PIM designs reveals a three-way tension between computational flexibility, memory proximity, and isolation guarantees. LiM systems excel in reducing data movement but are constrained in functionality and integration. NDP provides enhanced programmability but requires memory isolation enforcement and encrypted DMA channels to mitigate the risks introduced by offloading and runtime execution. NVM-based PIM demonstrates measured energy efficiency gains for in-memory AI acceleration but faces unresolved challenges in reliability and secure analog computing [35]. By situating future discussions in this taxonomy, supported by the comparative security posture shown in Table 1, we establish an evaluative basis for assessing existing security solutions and identifying threat vectors unique to each class of architecture.

Table 1. Architectural Taxonomy of PIM Design Classes: Strengths, Security Exposures, and Primary Threat Categories.

PIM Class	Representative Systems	Key Strengths	Security Exposures	Primary Threat Categories	Representative Attack/Reference	Mitigation Approaches
Logic-in-Memory (LiM)	Ambit [23]	Ultra-low data movement; efficient bulk bitwise operations	Row-level interference; predictable access patterns exploitable for RowHammer amplification; timing-based side-channels via non-standard row activation	Hardware faults (disturbance errors); timing side-channels via non-standard row activation sequences	RowHammer-style disturbance amplification via bulk activation patterns [24]	Refresh-based mitigation; row remapping [25]
Near-Data Processing (NDP)	Tesseract [26], UPMEM [10]	Programmable cores adjacent to DRAM; broader workload coverage	DMA exploits; co-tenant isolation failures; thermal coupling-induced retention faults; absence of built-in memory encryption	DMA-based privilege escalation; multi-tenant isolation leakage; hardware reliability faults	Cross-bank covert channel via shared DRAM substrate [36]	Memory isolation enforcement; encrypted DMA channels; hypervisor-level access controls
NVM-based PIM	PRIME [14] †	High density; analog in-memory compute; high energy efficiency for AI inference	Device variability and resistance drift enabling analog side-channels; wear-out patterns enabling backdoor insertion; fault leakage via resistance state corruption	ML weight inference via analog side-channels; long-term reliability manipulation; cross-layer fault attacks	Analog side-channel extraction of DNN model weights via resistance drift [15]	Device-level error correction; stochastic bit masking [15]

† NeuroSim [34] is a full-stack simulation framework, not a deployed PIM architecture; it is excluded from the Representative Systems column. The systems listed reflect the most widely cited research prototypes; none of the NVM-based PIM entries have confirmed commercial deployments as of this writing.

4. Security Landscape in Processing-in-Memory (PIM) Architectures

The security properties of Processing-in-Memory (PIM) are influenced by the host CPU, the operating system, device drivers, and the coherence protocols that govern data exchange. An adversary typically does not begin at the memory layer; attacks often

originate from privileged software running on the CPU or from malicious user applications that escalate privileges through the OS and subsequently issue unauthorized operations to PIM [17].

This layered interaction creates multiple trust boundaries that can be exploited. The host interface, typically implemented via PCIe, CXL, or custom coherence protocols, mediates communication between the CPU and PIM [36]. A compromised driver or malicious runtime can inject crafted PIM commands, manipulate memory mappings, or alter scheduling queues to cause unauthorized execution inside memory banks. Virtualization and resource-sharing mechanisms amplify risk: a hypervisor managing multiple tenants may unintentionally expose shared memory regions to cross-VM attacks, enabling adversaries to infer or corrupt sensitive data processed in PIM logic units [24].

The attack propagation path is cross-layer by nature. At the CPU level, an attacker can leverage speculative execution or cache-invisible DMA operations to mount coherence-based attacks that bypass OS-level protections. At the OS and firmware level, threats include privileged driver exploits, malicious kernel modules, and compromised runtime environments that interact directly with PIM through low-level instructions or memory-mapped registers. At the PIM layer itself, the adversary can trigger data-dependent execution patterns, exploit analog computation variability in non-volatile memory (NVM)-based designs, or launch classical DRAM attacks such as RowHammer that remain relevant when memory cells double as compute units [17]. Figure 3 illustrates this end-to-end attack surface; each layer boundary represents a semantic gap that adversaries exploit. The following subsections organize PIM-specific vulnerabilities into three structural levels: array-level faults, compute-logic-level weaknesses, and interface-level exposures.

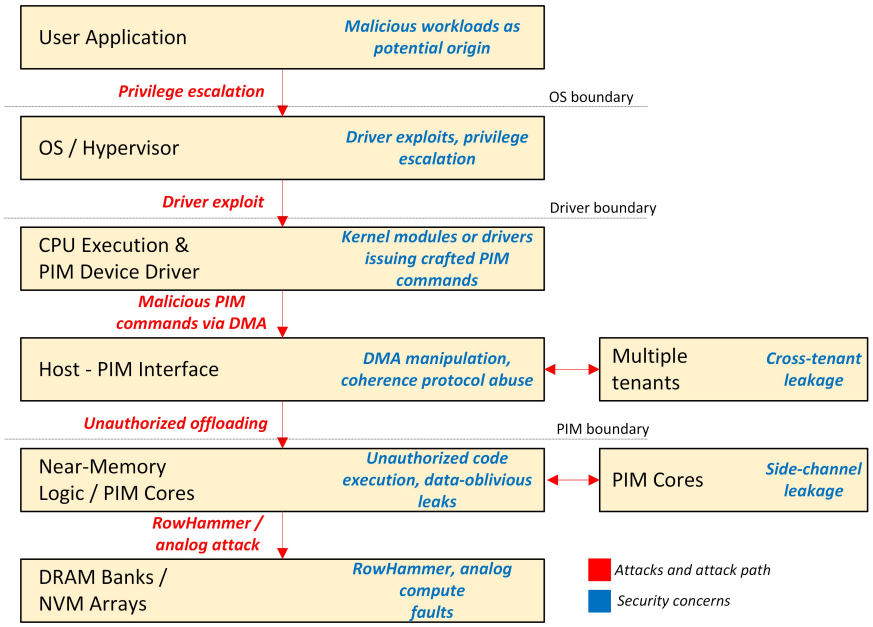


Figure 3. End-to-end attack surface of a PIM-enabled system. Red arrows denote attack paths; blue annotations identify the corresponding security concern at each layer. Attack propagation originates at the user application layer and traverses the OS boundary via privilege escalation, the driver boundary via kernel exploits, and the host–PIM interface via unauthorized DMA offloading, before reaching near-memory PIM logic units and the underlying DRAM or NVM array. The sidebar nodes illustrate lateral threats: cross-tenant leakage at the interface level and side-channel leakage at the PIM logic level. "PIM Cores" in the figure denotes the same component referred to as PIM logic units in the text, encompassing both LiM and NDP design classes as defined in Section 3.

4.1. Array-Level Faults

RowHammer represents a prominent array-level threat in PIM-enabled systems, arising from electrical interference between physically adjacent DRAM rows. The near-memory execution model exacerbates this vulnerability class because PIM logic units performing bandwidth-intensive workloads can generate access patterns structurally similar to adversarial hammering sequences. The mechanisms, access-pattern analysis, and proposed mitigations for RowHammer in PIM are examined in detail in Section 5.6. Data remanence constitutes a second array-level threat, arising from the residual persistence of data in NVM arrays after power-down or reset [37]. Unlike volatile DRAM, ReRAM, PCM, and MRAM technologies retain information unless deliberately erased, exposing cryptographic keys, intermediate machine learning results, and temporary inference data to post-shutdown recovery. In PIM architectures where compute operations are interleaved with storage at the bitline level, remanent data is distributed across subarrays, complicating comprehensive sanitization. The two adversary classes that exploit remanence, software-privileged attackers and physical attackers, together with architectural countermeasures, are examined in Section 5.5.

4.2. Compute-Logic-Level Weaknesses

At the compute-logic level, vulnerabilities arise in in-memory computation paths themselves. Since PIM logic units often operate on partially trusted or untrusted workloads, the absence of secure compute compartments becomes critical. Logic-in-memory (LiM) designs embed simple arithmetic units inside memory banks to accelerate operations like bitwise reductions or dot products. However, a survey of current LiM prototypes evaluated in the literature reveals that these units consistently lack support for privilege enforcement, control-flow validation, or access auditing [5]. Attackers can exploit this execution opacity to perform computation-injected attacks, embedding unauthorized logic or conditionals into data streams that trigger incorrect execution at runtime. In addition, programmable near-data processing (NDP) designs, which embed RISC-V or SIMD-class PIM logic units within memory stacks, are susceptible to classic code-reuse attacks, buffer overflows, or speculative execution vulnerabilities [38]. Critically, the absence of MMU support in current NDP-class PIM logic units means these isolation failures are structural rather than incidental: without hardware-enforced address-space separation, a compromised PIM logic unit has no architectural boundary preventing access to adjacent tenants' memory regions [26].

4.3. Interface-Level Exposures

At the interface level, host interface vulnerabilities represent the convergence point of traditional system security boundaries with the PIM execution model. As PIM devices increasingly support DMA (Direct Memory Access), message passing, and shared memory regions, the interfaces bridging host processors and memory become privileged channels subject to exploitation. A compromised OS or malicious driver may exploit semantic mismatches between host-side and memory-side logic, especially across PCIe or CXL coherent interconnects, causing PIM logic units to misinterpret descriptors and access unauthorized memory regions, thereby exposing systems to information leakage or memory corruption. The IMPACT study demonstrated how PIM-enabled covert and side-channel attacks leverage memory transaction latency differences to breach isolation, confirming that host and PIM logic can interpret shared data structures differently when no semantic validation layer exists [36]. Beyond this, the PIM-Enclave design explicitly addresses DMA integrity mismatches between host and PIM logic units by enforcing cryptographic

attestation over shared memory descriptors, providing a concrete architectural response to this class of vulnerability [39].

4.4. Multi-Tenancy and Isolation

Multi-tenancy and isolation concerns represent a significant challenge in data center or cloud scenarios where PIM-enabled memory is shared across virtualized or containerized workloads. Unlike traditional DRAM, where software-level isolation is mediated by page tables and TLBs, PIM introduces parallel compute elements that can access shared memory regions asynchronously. Without proper context separation or execution domain tagging, one tenant's PIM logic units could interfere with or spy on another tenant's data in shared subarrays or scratchpads. The challenge is further magnified when accelerators use bank-level parallelism, where multiple PIM logic units may compute simultaneously across overlapping regions. Several academic investigations, including [17], highlight fundamental gaps in current PIM architectures, particularly around the lack of address-space isolation, absence of MMU support, and opaque programming models that undermine secure multi-tenant execution. Although these proposals sketch hardware–software co-design approaches for tenant-aware scheduling and isolation, most remain at an experimental stage [40].

4.5. Interaction with OS Memory Management

In conventional systems, the operating system's memory management unit (MMU) and hypervisor-managed extended page tables (EPTs) enforce address-space isolation between tenants. PIM complicates this model because PIM logic units execute memory-side operations that bypass the host MMU entirely: a PIM core issuing row activations operates on physical addresses within its local bank, outside the scope of host-managed virtual-to-physical translation. The proposed hardware-level enforcement (Section 7) must therefore interact with host-side memory management at two coordination points. First, during dynamic memory allocation and cross-tenant page migration, the hypervisor must propagate tenant-identity updates to the per-subarray access-control metadata maintained by the PIM hardware; failure to synchronize these updates creates a window during which a migrated page retains the prior tenant's access tag, enabling unauthorized access. Second, during DMA offloading and PIM kernel scheduling, the host OS scheduler must communicate tenant context identifiers to the PIM controller so that bank-level isolation tags are bound to the correct execution context before any PIM operation proceeds. Siloz [41] demonstrates that subarray-group isolation can be enforced with negligible performance overhead ($\pm$0.5% of baseline) when the hypervisor correctly manages EPT integrity, providing evidence that hardware–software coordination at this boundary is feasible. The outstanding challenge is that no current PIM architecture exposes an interface through which the host MMU can programmatically update memory-side access-control metadata, making this coordination point a concrete standardization requirement (consolidated in Section 7.3).

As PIM transitions from experimental labs to production systems, securing it will require a fundamental re-evaluation of execution privileges, fault tolerance, and overall system co-design. The subsequent sections examine proposed defenses, analyze their effectiveness, and propose a cohesive framework for securing PIM at scale.

5. Review of Existing Security Mechanisms in PIM Systems

Most existing mechanisms either adapt legacy CPU-centric techniques or are limited to academic prototypes that address narrow attack vectors under idealized assumptions. In this section, we critically examine the state of security mechanisms for PIM, organized by defense category, and identify the limitations in threat coverage, trust as-

sumptions, and scalability that restrict applicability in multi-tenant and heterogeneous computing environments.

5.1. Host–PIM Orchestration and Iago-Style Threats

Host–device orchestration remains a critical yet vulnerable component of Processing-in-Memory (PIM) security. In typical deployments, the host operating system or hypervisor manages kernel scheduling, PIM program loading, and memory buffer allocation. This centralization places significant trust in the host by design: all PIM execution paths flow through host-controlled data structures. A compromised host therefore enables Iago-style attacks [42], in which a malicious kernel supplies adversarially crafted return values or memory descriptors to subvert the execution of an otherwise protected PIM process [43].

The Ahn et al. PIM-Enabled Instructions architecture [44] illustrates how this dependency arises at the instruction level: because PIM offload decisions pass through host-managed instruction streams, a compromised host can alter those streams or manipulate memory descriptors to violate integrity guarantees. Partial mitigations such as capability-based addressing, tag-based access control, and memory-region firewalls offer some protection. For example, UPMEM's architecture confines each DPI thread to a restricted memory segment by design [10], providing a degree of cross-thread address isolation, though this has not been evaluated as a security boundary under adversarial conditions. These measures cannot fully prevent pointer manipulation or controlled boundary violations originating from a compromised host.

CXL-attached memory expands the threat environment considerably [45]. Unlike PCIe, which provides non-coherent point-to-point transport, CXL adds cache-coherent protocols (CXL.cache and CXL.mem) that transmit instructions and memory descriptors in formats that often lack strong cryptographic integrity checks or replay protections [45]. Adversaries can therefore inject stale, reordered, or forged commands to coerce unauthorized PIM operations [46]. DEV-PIM addresses this by introducing a hardware-assisted orchestration validation framework [47]. The PIM memory controller incorporates runtime verification logic that checks instruction sequences and memory requests against secure metadata stored in DRAM. Lightweight cryptographic hashes and sequence numbers detect stale or forged commands. Evaluated on FPGA and simulation platforms against data-analytics benchmarks including graph traversal and database scan workloads, DEV-PIM imposes under 2% performance overhead, primarily from verification cycles, while significantly raising the cost of Iago-style exploits [47].

These frameworks advance host-independent verification and cryptographic attestation that reduce Iago-style vulnerabilities by removing implicit trust in the host. However, challenges remain. Increased metadata storage, cryptographic computation, and hardware complexity must be carefully balanced against PIM's parallelism, low-latency requirements, and energy efficiency. Extending DEV-PIM's co-design discipline [47] to heterogeneous and multi-tenant PIM deployments requires two capabilities that no current proposal provides: instruction-sequence verification that scales across CXL-attached devices with independent controllers, and attestation protocols that do not assume a single trusted host. Until both are available, host-independent orchestration security cannot be claimed for production CXL-scale deployments.

5.2. Trusted Execution Environments (TEEs) in PIM Cores

A parallel line of work addresses the trust problem by embedding enclave-like isolation directly within PIM execution engines, drawing on CPU and accelerator TEE designs including Intel SGX, ARM TrustZone, and RISC-V frameworks such as Keystone and Sanctorum [48–51]. The premise is consistent across this body of work: because the host

cannot be trusted, attestation boundaries must be relocated toward the memory substrate. The difficulty is that each deployed design relocates those boundaries by one architectural level while leaving the root of trust anchored at the host, a compromise that PIM's physical disaggregation structurally undermines.

SE-PIM partitions sensitive computation using MPC-style secret sharing between a CPU TEE and untrusted UPMEM PIM hardware [39,52]. Encrypted data shares are distributed such that computation-intensive operations execute on the PIM while the CPU TEE handles lighter tasks, with pre-computation moving expensive MPC operations offline. Evaluated on production UPMEM DIMMs, SE-PIM achieves a 14–15× speedup over a CPU-only secure baseline on DLRM and MLP inference workloads. SecNDP applies the same partitioning logic at a coarser offload boundary, reporting 7.5× speedup and 18% energy reduction over a secure non-NDP baseline on linear workloads [53]. Both evaluations are conducted on hardware or validated simulation, and the reported gains reflect genuine reductions in secure computation overhead relative to CPU-centric baselines.

Neither design, however, escapes the constraint that a single host-side TEE coordinates all attestation. Toleo makes this limitation most apparent: it uses PIM-enabled smart memory over a CXL IDE network to replace the Merkle tree that processor TEEs require for freshness guarantees, allowing a single 168 GB device to cover replay-attack protection across a 28 TB CXL-expanded pool [54]. This is a substantive advance in freshness scalability. The attestation path, however, remains the CXL IDE channel between the smart memory device and the host controller. In a vendor-disaggregated deployment, where the smart memory is sourced from one manufacturer and the host controller from another, no published mechanism establishes the trustworthiness of that channel without host involvement. The scalability gain does not resolve the structural dependence.

The common architectural failure across SE-PIM, SecNDP, and Toleo is that each correctly moves computation toward memory while retaining a host-anchored attestation hierarchy that PIM's disaggregated topology undermines. This dependence has a concrete security consequence: timing and power side-channels arising from shared refresh and sense-amplifier circuitry are not observable by a host-anchored enclave validator, as IMPACT demonstrates quantitatively [36]. Closing this gap requires relocating the root of trust into the memory substrate itself, such that host software obtains attestation from memory-resident verifiers rather than asserting it over memory. The distributed per-bank trust-anchor model proposed in Section 7 is one architectural approach to this; its feasibility within HBM2e timing constraints is an open empirical question, but the structural argument for the approach follows from the analysis of this section.

5.3. Resource Isolation and Scheduling Techniques

Multi-tenant PIM deployments face a physical isolation constraint that software-layer partitioning cannot fully address: DRAM sense amplifiers and refresh circuitry are shared across tenants at a granularity below any logical address-space boundary. Proposals in this area fall into two categories, static spatial partitioning and dynamic scheduling, with different security properties and the same physical ceiling.

Static partitioning approaches, of which NeuroPIM is representative, assign a single neural network per vault in an HBM-based 3D-stacked architecture and impose priority-based request multiplexing to give PIM traffic precedence in the address queue [55]. The architecture achieves a measured 17.8× average speedup and 88% energy reduction relative to PU-based execution on its evaluated workloads, with peaks of 46× on specific benchmarks. The security inference, that vault-level assignment constrains cross-tenant interference, follows from the addressing boundary: under correct operation, one tenant's PIM thread cannot issue memory operations into another tenant's vault address range.

This inference is not evaluated adversarially in [55]. Sense amplifiers and refresh operations in HMC span vault boundaries; a co-located adversary with timing visibility can observe refresh-induced latency variation from outside a victim's vault address range, a threat that vault assignment does not constrain. Static partitioning of this form provides deterministic fault containment, but cannot adapt to shifting workload profiles and creates contention at partition boundaries that, in DRAM-based systems, is exploitable for cross-tenant RowHammer attacks under hypervisor-enforced isolation [41].

Dynamic scheduling, represented by Virtual PIM, implements a software runtime that monitors DPU utilization and migrates tasks across UPMEM DPUs without hardware modification to PIM controllers or memory [56]. Evaluated on production UPMEM hardware across database scan and neural network inference workloads, it achieves up to 2.3× throughput improvement over static DPU allocation with improved load distribution. Dynamic reallocation reduces the duration of co-residency between tenants on a given DPU, structurally narrowing the window available to contention-based timing attacks. This is the authors' inference from the scheduling behavior; co-residency duration and timing-channel exposure are not measured adversarially in [56]. Task migration does not alter the physical coupling: shared refresh circuitry and sense amplifiers within DRAM subarrays persist regardless of logical task placement, as established by the DRAMA measurements [57]. Furthermore, task migration incurs a non-trivial context-switch latency penalty that Virtual PIM does not quantify adversarially. Each migration requires flushing the source DPU's working set (scratchpad contents, register state, and in-flight memory requests) to DRAM, transferring context metadata to the destination DPU, and reloading the working set from DRAM into the destination DPU's scratchpad. On UPMEM hardware, the DPU scratchpad (WRAM) is 64 KB per DPU; at the measured WRAM-MRAM transfer bandwidth of approximately 800 MB/s [58], a full scratchpad flush-and-reload cycle requires approximately 160 µs, during which the migrating task cannot execute and the destination DPU is unavailable to other tenants. For latency-sensitive workloads, this migration overhead imposes a lower bound on the co-residency window that dynamic scheduling can achieve: migrations cannot occur more frequently than once per ∼160 µs without the migration overhead itself exceeding the useful computation time. Establishing acceptable migration overhead bounds, defined as the maximum fraction of epoch time that can be consumed by migration without negating PIM's throughput advantage, requires workload-specific characterization that the published literature does not yet provide.

Static partitioning and dynamic scheduling therefore represent complementary design points rather than competing solutions: the former establishes hard spatial boundaries at the cost of utilization flexibility; the latter reduces adversarial co-residency windows at the cost of spatial determinism. Neither eliminates physical-layer side channels without additional hardware enforcement. The mechanism that would close the remaining gap, hardware-enforced bank-level access quotas with per-tenant refresh isolation operating independently of task placement, requires modifications to DRAM controller micro-architecture that no commodity PIM device currently supports, and its latency implications for disaggregated deployments have not been characterized in the published literature.

5.4. Information Leakage and Side-Channel Mitigations

Resource sharing in PIM systems produces three structurally distinct leakage classes, each of which requires a separate adversary model and mitigation strategy. The first is row-buffer timing leakage, where a co-located unprivileged process exploits the timing differences between DRAM row hits and row conflicts to establish covert channels or infer victim access patterns. The second is memory-bus address leakage, where an adversary with physical or logical visibility of the memory bus observes address streams to reconstruct

the workload behavior. The third is on-DIMM microarchitectural leakage in NVM systems, where an attacker sharing NVM cache structures or wear-leveling state recovers execution paths or stored data. Proposals that address one class in isolation routinely leave the other two unmitigated; the distinction is not merely taxonomic but determines which threat model and hardware assumptions a proposed defense requires.

5.4.1. Row-Buffer Timing Leakage

IMPACT quantifies the severity of this channel in PIM-enabled systems [36]. By exploiting the shared DRAM row buffer through PIM-enabled direct memory access, an unprivileged co-located process bypasses the cache-bypass overhead that processor-centric timing attacks require and leverages PIM's intrinsic bank-level parallelism. Measured covert channel throughput reaches 8.2 Mb/s and 14.8 Mb/s for processing-near-memory and processing-using-memory variants respectively, 3.6× and 6.5× above the prior state of the art for main-memory covert channels, with a companion side-channel recovering private genomic query characteristics at 7.6 Mb/s and 96% accuracy. IMPACT evaluates two candidate defenses: bank-level memory partitioning, which eliminates the shared row-buffer state at the cost of limiting concurrency to the number of physical banks, and PIM instruction throttling, which reduces channel throughput at substantial performance cost. Both incur overhead that conflicts with PIM's bandwidth and latency requirements. The authors' conclusion that low-overhead mitigation for this attack class is an open problem, is supported by the measurement data and has not been contradicted by subsequent published work.

5.4.2. Memory-Bus Address Leakage

InvisiMem addresses the distinct threat of an adversary monitoring the external memory bus [59]. A modified Micron Automata Processor prototype decrypts memory addresses internally before they reach the DRAM array, preventing address-stream observation on the bus, while constant-rate heartbeat packets enforce uniform bus activity to defeat traffic-analysis timing attacks. Compared to ORAM-based alternatives, InvisiMem reduces overhead by one to two orders of magnitude across performance, space, energy, and bandwidth dimensions, at an overall cost of approximately 20.74% performance overhead and 37.5% memory space overhead from heartbeat injection. InvisiMem is a complete solution for its defined adversary model: it eliminates address visibility on the bus. It does not address row-buffer timing variation within the DRAM array, which requires a distinct mitigation. PIM-ORAM extends coverage to access-pattern leakage more broadly via a split-data ORAM scheme retrofitted to commodity UPMEM hardware, providing ORAM's formal indistinguishability guarantee while exploiting PIM parallelism to reduce the latency overhead that renders conventional ORAM impractical [60].

5.4.3. On-DIMM Microarchitectural Leakage in NVM Systems

NVLeak establishes that on-DIMM cache structures and wear-levelling policies in Intel Optane DIMMs support cross-core and cross-VM covert channels as well as cache-based side channels that defeat defenses targeting on-chip resources exclusively [61]. The adversary shares NVM cache sets with the victim and requires no elevated privileges. Cache-set partitioning that prevents NVCache sharing between tenants reduces the measured overhead to below 4%. PME addresses the complementary leakage surface at the array level: PIM-based masking and encoding applied within the NVM array suppresses power and data-dependent side-channel signals at the point of computation, targeting the supply-current leakage that cache partitioning alone cannot eliminate [62]. Together, NVLeak and PME cover the two primary leakage surfaces in NVM-based PIM systems.

Both defenses are evaluated on single-DIMM configurations; their composition across multi-device racks under adversarial scheduling has not been demonstrated.

The shared architectural limitation across InvisiMem, PIM-ORAM, and NVLeak is deployment scope: each enforces a uniform leakage policy within a single memory controller's authority. Disaggregated PIM hierarchies provide no such authority boundary, and no published work demonstrates consistent leakage-policy enforcement across two physically separate controllers under adversarial conditions. This is not an engineering gap that incremental optimization closes; it requires a fundamentally different enforcement model in which leakage policy is co-located with computation rather than imposed from a central controller.

5.5. Data Remanence and Secure Erasure Mechanisms

Data remanence in PIM systems presents three structurally distinct threat scenarios whose mitigations are not interchangeable. In the first, a post-power-off adversary with DIMM-level physical access recovers plaintext from NVM cells that retain state without power [63]. In the second, a privileged attacker or physical adversary exploits inconsistent persistence of encrypted data and integrity metadata across a power failure to recover or corrupt sensitive state [64]. In the third, sensitive intermediate values, neural network weights, activation buffers, partial computation results, remain accessible in NVM arrays between workload phases because no just-in-time erasure mechanism operates at the PIM execution layer [65]. Conventional architectures address the first two scenarios through zeroization primitives and crash-consistent encryption schemes that depend on centralized orchestration and synchronous host commands. PIM computation distributes transient state across subarrays, bitlines, and in-flight buffers, complicating both identification and purging of security-sensitive residues; the third scenario has no deployed solution.

5.5.1. Physical Remanence

Silent Shredder eliminates the physical remanence window by repurposing counter-mode encryption initialization vectors to perform shredding operations, avoiding the write-amplification penalty that conventional zeroization imposes on NVM endurance [63]. Evaluated on gem5 with PowerGraph and 26 SPEC 2006 workloads, it reduces initialization-phase writes by an average of 48.6%, accelerates shredded cache-line reads by 3.3×, and improves IPC by 6.4% on average. The design principle it establishes, repurposing existing cryptographic state to perform erasure at zero write cost, is the appropriate foundation for PIM-aware erasure schemes. Silent Shredder's architecture, however, assumes a centralized memory controller with exclusive ownership of the encryption engine. In PIM deployments, transient computation state is distributed across subarrays whose in-flight buffers and per-core scratchpads the central controller cannot enumerate; the counter-repurposing mechanism has no reach into those structures.

5.5.2. Crash-Recovery Leakage

SecPM integrates counter-mode encryption with crash-consistent persistence through a counter cache write-through scheme and locality-aware counter write reduction, achieving up to 93% counter write reduction and 1.3–2.0× transaction speedup over a baseline secure NVM configuration [64]. Triad-NVM extends crash-recovery security through hierarchical Merkle Tree integrity verification, reducing recovery time from over 30 s to under 4 s in a gem5-simulated 8 TB NVM configuration while doubling throughput over strict-persistence models [66]. SecPB advances this further by moving the point of secure persistency toward the processor core through battery-backed persistent structures, reducing performance overhead by up to 32.8× relative to a strict-persistency baseline [67]. These three designs collectively establish that crash-recovery leakage is tractable for cen-

tralized NVM controllers. None defines a mechanism for identifying which in-flight PIM computation intermediates are security-sensitive, because all three assume an execution model in which computation and storage occupy distinct physical layers, an assumption that PIM's bitline-level co-location invalidates.

5.5.3. Wear-Leveling Interaction with Secure Erasure

An additional complication arises from wear-leveling algorithms native to NVM controllers. Flash translation layers (FTLs) and wear-leveling policies remap logical addresses to physical locations transparently to the host, ensuring uniform write distribution across the NVM array to maximize device lifetime. This remapping directly undermines just-in-time decryption and secure erasure mechanisms: when Silent Shredder or SFGE issues an erasure command targeting a logical address, the wear-leveling controller may have already remapped the underlying physical cells, leaving stale copies of sensitive data at the original physical location. The erasure command operates on the current logical-to-physical mapping, not on all historical physical locations that previously held the data. Bypassing the wear-leveling controller to perform direct physical erasure requires either (i) exposing the internal address mapping table to the security layer, which breaks the abstraction boundary that enables wear-leveling optimization, or (ii) implementing secure erasure below the FTL within the NVM array controller itself, which requires modifications to proprietary NVM controller firmware. Neither approach has been demonstrated in a PIM context. For PIM-specific deployments, the interaction is further complicated by the fact that PIM computation generates write patterns that the FTL was not designed to anticipate: compute-generated intermediate writes may trigger wear-leveling remaps during active computation, scattering security-sensitive intermediates across physical locations that no subsequent erasure command can enumerate. Addressing this requires either PIM-aware wear-leveling policies that track security-sensitive pages separately, or cryptographic approaches (such as Silent Shredder's counter-repurposing) that render physical location irrelevant by ensuring data is always encrypted with ephemeral keys that are destroyed on erasure. The implications for the proposed framework are consolidated in Section 7.3.

5.5.4. Compute-Layer Remanence

SFGE addresses compute-layer remanence for neural network weights through a runtime encryption scheduling scheme that applies just-in-time decryption and immediate re-encryption to the weight parameters with the largest gradient magnitudes, achieving model obfuscation at lower performance overhead than full-weight encryption [65]. The original paper does not report a quantified wear-reduction figure against a measured baseline; the efficiency claim is, therefore, qualitative. More consequentially, SFGE operates at the model level within a single workload type and does not address arbitrary intermediate values such as activation buffers, partial sums, and scratchpad contents, which represent the primary remanence exposure in multi-tenant PIM deployments where workloads from distinct tenants occupy the same physical arrays across successive scheduling epochs.

Two design problems remain unresolved in the published literature. First, no formal model exists for identifying security-sensitive intermediates within a distributed PIM execution: Silent Shredder and SecPM consider a centralized controller capable of enumerating all sensitive state, and PIM's disaggregated execution model removes that precondition without providing a replacement. Second, tamper-proof trigger conditions for in-array erasure require attestation of the trigger logic itself, a dependency that reduces to the trust-anchor problem examined in Section 5.2. Progress on compute-layer remanence is therefore contingent on progress in distributed attestation; the two problems are structurally coupled and not independent research directions.

5.6. RowHammer and Memory Reliability Defenses

RowHammer remains one of the most persistent reliability and security challenges in DRAM-based systems [68]. Repeated activation of specific rows induces electrical coupling and charge leakage that corrupts neighboring cells through unintended bit flips. Conventional mitigations such as Target Row Refresh (TRR), Error-Correcting Codes (ECC), and refresh-rate tuning operate under CPU-managed memory models, where access scheduling and refresh operations are centrally orchestrated [69,70]. Processing-in-Memory (PIM) architectures alter this control structure: compute units embedded within DRAM banks generate their own row activation sequences independently of the host, and the host-managed refresh controller has no visibility into those internal patterns. This removes the assumption on which host-centric mitigations depend.

PIM introduces two RowHammer exposure scenarios that conventional mitigations cannot address. In the first, a benign but memory-intensive PIM kernel inadvertently generates high-frequency row activations that exceed the RowHammer threshold within a bank, producing reliability failures without attacker involvement. In the second, the authors' structural analysis of the attack surface, for which no published PIM-specific exploit currently exists, a malicious PIM kernel could deliberately issue dense activation sequences targeting adjacent rows, exploiting the absence of host-level rate limiting on PIM-internal operations to induce bit flips in a co-tenant's memory region. Blacksmith establishes the severity of the underlying vulnerability from the CPU side: non-uniform, frequency-modulated hammering patterns bypass TRR on all 40 DDR4 DIMMs in its test pool, generating on average 87× more bit flips than prior uniform patterns [11]. The central finding of Blacksmith that TRR's row-sampling logic fails against non-uniform activation sequences, applies structurally to PIM, whose compute units generate workload-dependent activation patterns that TRR's sampler cannot observe or anticipate, for the same reason that CPU-side non-uniform patterns evade it.

To address these exposures, LT-PIM and P-PIM embed RowHammer awareness directly within DRAM subarrays, removing the dependency on host-managed refresh entirely [12,13]. LT-PIM integrates LUT-based arithmetic queries alongside charge-sharing bitwise logic through reconfigurable sense amplifiers, exploiting this in-DRAM compute capability to enable periodic XNOR integrity checks on LUT rows without external counters, SRAM/CAM resources, or host involvement [12]. Validated using a cross-layer framework combining Cadence Spectre, Synopsys Design Compiler, modified CACTI, and gem5, LT-PIM imposes a worst-case 0.14–0.2% execution slowdown and achieves up to 80% energy savings over prior RowHammer tracking frameworks, at the cost of dedicated LUT and compute rows. P-PIM extends this design lineage to dual-row and triple-row activation for higher-throughput bitwise parallelism across wider operand configurations [13]. Its RowHammer protection mechanism applies the same periodic XNOR check infrastructure, imposing 0.4–0.8% worst-case slowdown and achieving 71% energy savings over SRAM/CAM-based tracking frameworks. The two sets of overhead figures are not directly commensurable (P-PIM's evaluation omits Synopsys Design Compiler relative to LT-PIM) and should be treated as indicative of the design space rather than as a controlled performance comparison.

Both designs embed tracking within the compute infrastructure, which is the correct architectural approach for removing host dependency. The open question is whether periodic integrity checks are sufficient against non-uniform activation patterns. Blacksmith demonstrates that frequency-modulated hammering sequences bypass TRR on all 40 DDR4 DIMMs in its test pool, generating 87× more bit flips than uniform patterns by exploiting the fixed sampling logic of TRR's row-activation counter [11]. A periodic XNOR check operates on a fixed schedule, not on activation events; a sufficiently informed adversary can

concentrate dense hammering sequences within the intervals between scheduled checks, evading detection by the same mechanism that Blacksmith exploits against TRR. LT-PIM and P-PIM do not evaluate this attack surface, and the hammering threshold below which purely local, schedule-based detection is reliably safe against non-uniform patterns is not established in the published literature.

Defending against RowHammer in disaggregated PIM configurations requires resolving two distinct problems that the surveyed literature has not addressed simultaneously. The first is coordinating activation counts across banks that share no refresh controller, the configuration that CXL-attached PIM creates, without introducing synchronization latency that undermines PIM's bandwidth advantage. The second is establishing whether any schedule-based in-array tracking mechanism provides meaningful security guarantees against an adversary with knowledge of, or the ability to infer, the check schedule. Progress on the second problem is a prerequisite for evaluating proposals that address the first.

5.7. Synthesis: Fragmented Defenses, Missing Convergence, and Underspecified Security Budgets

Across the surveyed literature, Processing-in-Memory (PIM) security mechanisms evolve along narrow axes, host–PIM orchestration, side-channel resilience, isolation, remanence, and reliability, without converging into a unified, composable defense stack. These efforts address narrow threat axes in isolation, each tied to specific platform assumptions and workload types. This lack of convergence complicates composition, particularly in heterogeneous, cloud-scale deployments where trust boundaries and interference patterns shift dynamically. Table 2 consolidates these dimensions across representative proposals.

This fragmentation becomes more acute when examining the underlying performance–security budgets. PIM security mechanisms consistently balance meaningful protection against modest performance and hardware costs, yet quantitative accounting of these overheads remains incomplete and inconsistent across the literature. The surveyed systems do not report area overhead in a uniform way, so no single cross-paper figure is available; where die area is reported, it ranges from under 1% for in-DRAM logic additions such as P-PIM's dual-row activation rows [13] to several percent for more complex controller augmentations such as DEV-PIM's runtime verification logic [47]. Performance overheads span a wide band: DEV-PIM reports under 2% slowdown on data-analytics workloads [47]; LT-PIM and P-PIM report RowHammer self-tracking overheads of 0.14–0.8% [12,13]; InvisiMem reports approximately 20.74% performance overhead for memory bus side-channel protection [59]; and IMPACT's authors note that their evaluated bank-level partitioning defense imposes overhead sufficient to render low-overhead mitigation an open problem [36]. Latency increases of only a few cycles per operation, typically under 10 ns, are achievable for embedded checks. However, the interaction of these microarchitectural delays with bank-level parallelism and contention patterns can amplify both throughput degradation and side-channel leakage, creating a direct trade-off that no current proposal quantifies end-to-end across a full defense stack.

Bandwidth overhead is a critical and frequently understated dimension. In-memory cryptographic and integrity operations introduce metadata traffic that competes directly with user data flows. InvisiMem reports 37.5% memory space overhead from heart-beat packet injection and address encryption [59]. In secure non-volatile memory (NVM) systems, counter-mode encryption requires a counter write for every data eviction; without optimization, this counter traffic alone adds significant write amplification. SecPM's locality-aware counter write reduction (CWR) addresses this by exploiting spatial locality, reducing counter writes by up to 50% on its evaluated workloads [64]. Triad-NVM's selective persistence policy halves integrity-tree write traffic relative to full strict persistence [66]. Even after these optimizations, the residual counter and Merkle Tree metadata traffic in

write-intensive workloads occupies bandwidth that directly competes with user data and can amplify timing-channel exposure under contention. The Table 2 entry for SecPM is blank because the original paper does not report a standalone performance overhead figure relative to an unprotected NVM baseline [64]. The literature lacks systematic frameworks that quantify how area, timing, and bandwidth costs aggregate across defense stacks covering multiple threat classes simultaneously.

Table 2. Comparative Summary of PIM Security Mechanisms. Performance overhead figures are not directly comparable across entries; each reflects a different evaluation platform, workload suite, and baseline configuration. Readers should consult individual papers for methodology.

Work	Threat	Eval. Basis	Security Metrics	Perf. Overhead	Platform	Limitations
DEV-PIM [47]	Iago-style orchestration	FPGA + sim.	Prevents forged/stale command execution	0.35% CPU overhead	DRAM-PIM, FPGA	Metadata overhead; verification vs. throughput trade-off
Ghinani et al. [71]	Malicious host; PIM offload leakage	Real UPMEM HW	Confidentiality/integrity via MPC secret sharing	14.66× speedup over CPU-Secure baseline	UPMEM (2496 DPUs)	Precomputation required; key management complexity
SE-PIM [52]	Untrusted PIM core leakage; address side-channel	Real UPMEM + sim.	Side-channel resistance; isolation; integrity	14–15× speedup over CPU-only TEE	UPMEM DRAM PIM	Static trust boundaries; AES acceleration required
SecNDP [53]	NDP offload leakage to untrusted memory	Sim. + emulation	Confidentiality; integrity via linear checksum	7.46× speedup; 18% energy reduction	DRAM NDP, multi-core	Linear ops only; overhead on nonlinear workloads
Toleo [54]	Replay/freshness on CXL memory	Sim. (graph, AI, DB)	Per-block freshness across 28 TB; version confidentiality	2% overhead; 98% version cache hit rate	CXL smart memory, 168 GB	No RowHammer coverage across disaggregated banks
IMPACT [36]	Covert/timing channel via shared row buffer	Sim., DRAM-PIM	8.2–14.8 Mb/s covert; 96% side-channel accuracy	Bank partitioning: high (open problem)	DRAM PIM arrays	No low-overhead mitigation; systemic defense required
InvisiMem [59]	Memory bus address/timing leakage	Sim., HMC smart mem.	Address-stream leakage eliminated; uniform bus activity	∼20.74% perf.; 37.5% space overhead	HMC smart DRAM	Smart memory required; not for commodity DDRx
PIM-ORAM [60]	Access-pattern leakage	Real UPMEM HW	ORAM indistinguishability guarantee	Lower than conv. ORAM via PIM parallelism	UPMEM DRAM PIM	Central controller assumed; row-buffer channels unaddressed
SecPM [64]	Crash-recovery leakage; counter inconsistency	gem5 + NVMain (PCM)	Atomic counter durability; 93% counter write reduction	Not reported vs. unprotected baseline	NVM-PIM (PCM)	Physical remanence unaddressed; residual counter bandwidth
Triad-NVM [66]	Rollback/replay; crash-inconsistent metadata	gem5, 8 TB NVM	Selective Merkle persistence; recovery >30 s → <4 s	4.9% vs. strict persistence; 2× throughput	NVM-PIM	Physical remanence unaddressed; metadata overhead under writes
SecPB [67]	Crash-recovery leakage; metadata inconsistency	Sim., persistent mem.	Crash recoverability with lazy metadata generation	32.8× reduction vs. strict-persistency	NVM persistent mem.	Not validated in disaggregated PIM deployments
Silent Shredder [63]	Physical remanence; shredding write amplification	gem5 (PowerGraph, SPEC 2006)	Zero-write-cost shredding; plaintext eliminated	48.6% write reduction; 3.3× read speedup; 6.4% IPC gain	NVM controller	Centralized only; no reach into distributed PIM scratchpads
SFGE [65]	Compute-layer remanence of NN weights	NN benchmarks (DLRM, MLP)	Model obfuscation via just-in-time weight encryption	Negligible latency; wear reduction not quantified	NVM-PIM (ML)	Weights only; does not cover arbitrary PIM intermediates
Virtual PIM [56]	Co-tenant contention; cross-tenant DPU sharing	Real UPMEM HW	2.3× throughput over static allocation	4–7% vs. static DPU allocation	UPMEM, SW runtime	Physical-layer side channels persist (shared refresh/sense amps)
NeuroPIM [55]	Static tenant isolation for NN workloads	HW-accelerated PIM sim.	Vault-level containment; 17.8× speedup; 88% energy reduction	Not reported; no adversarial eval.	HMC 3D-stacked DRAM	Inflexible under dynamic workloads; no adversarial evaluation
P-PIM [13]	RowHammer from in-DRAM activations	Spectre, CACTI, gem5	In-DRAM XNOR self-tracking; no external counters	0.4–0.8% slowdown; 71% energy saving vs. SRAM/CAM	Programmable DRAM	Higher overhead than LT-PIM; modified row decoders required
LT-PIM [12]	RowHammer; anomalous access in LUT DRAM	Spectre, Synopsys DC, CACTI, gem5	Periodic XNOR on LUT rows; no external counters	0.14–0.2% slowdown; 80% energy saving vs. prior frameworks	DRAM PIM, LUT arith., reconf. sense amps	May miss non-uniform hammering; dedicated LUT rows required

For cross-mechanism comparison despite heterogeneous evaluation platforms, Table 3 summarizes reported overhead dimensions for each defense category. Values are drawn directly from the original papers; entries marked "—" indicate that the dimension was not

reported. Readers should note that these figures are not directly commensurable: each reflects a different evaluation platform, baseline configuration, and workload suite. The table is intended to convey the order-of-magnitude cost landscape across defense categories rather than to support precise cross-paper ranking.

At the software layer, the compiler and programming model are tightly coupled to these hardware and timing trade-offs but are rarely treated as explicit parts of the security design. Current PIM compilers and SDKs focus on tiling, placement, and scheduling to exploit bank-level parallelism and reduce data movement, with only incidental consideration of how transformations affect side-channel leakage or isolation boundaries. This mirrors long-standing issues in CPU systems, where aggressive optimizations can remove security-critical operations or reintroduce data-dependent control flow. In PIM the consequences are more severe, because the toolchain decides which tenants share banks, subarrays, and row buffers. Enclave-style designs such as SE-PIM and SecNDP rely on secure APIs and handles to delineate protected regions, yet their guarantees depend on compilers preserving oblivious access patterns and avoiding unprotected temporaries. Remanence-oriented systems such as SecPM and Triad-NVM assume that erase and persist operations act as semantic barriers that must not be reordered or eliminated. Isolation schemes such as Virtual PIM and NeuroPIM implicitly trust runtime and compiler placement to respect partitioning policies, even as IMPACT demonstrates that adversarial workloads can exploit exactly those mapping and scheduling freedoms to establish high-bandwidth covert channels [36]. PIM security cannot be reasoned about purely at the hardware level. It requires security-aware compilers and runtimes that expose annotations for partitioning, obliviousness, and data lifecycles, treat these as hard constraints during optimization, and verify that high-level security properties survive lowering into memory-resident kernels.

Table 3. Reported Overhead Comparison Across PIM Security Mechanisms.

Defense Category	Representative Work	Performance Overhead	Area Overhead	Bandwidth Overhead	Energy Impact	Eval. Platform
Orchestration verification	DEV-PIM [47]	0.35% CPU	Controller logic	Negligible	—	FPGA + sim.
TEE/MPC partitioning	SE-PIM [52]	14–15× speedup [a]	AES engine	MPC comm. overhead	—	UPMEM HW
Resource isolation (static)	NeuroPIM [55]	Not reported	Vault-level logic	Negligible	88% reduction	HMC sim.
Resource isolation (dynamic)	Virtual PIM [56]	4–7%	None (SW only)	Migration traffic	—	UPMEM HW
Bus side-channel	InvisiMem [59]	∼20.74%	Crypto engine	37.5% space	—	HMC sim.
Access-pattern (ORAM)	PIM-ORAM [60]	Lower than conv. ORAM	—	Log. overhead	—	UPMEM HW
RowHammer tracking	LT-PIM [12]	0.14–0.2%	Dedicated LUT rows	Negligible	80% saving [b]	Cross-layer sim.
RowHammer tracking	P-PIM [13]	0.4–0.8%	Modified row decoders	Negligible	71% saving [b]	Spectre + gem5
Secure erasure	Silent Shredder [63]	6.4% IPC gain	Counter storage	48.6% write reduction	—	gem5
Crash recovery	SecPM [64]	—	Counter cache	50% counter write reduction	—	gem5 + NVMain

[a] Reported as speedup over a CPU-only secure baseline, not as overhead. [b] Energy savings relative to prior SRAM/CAM-based RowHammer tracking frameworks, not overall system energy.

These limitations become most pronounced in multi-tenant environments where threats span host firmware, PCIe/CXL semantics, controller microcode, and subarray-level failure modes. Leakage controls such as InvisiMem and PIM-ORAM rely on trusted orchestration [59,60]. Enclave-based protections such as SE-PIM and Toleo depend on refresh integrity and disturbance-free scheduling [39,54]. Isolation frameworks such as Virtual PIM assume deterministic compiler placement [56]. Under adversarial scheduling or cross-tenant interference, these assumptions degrade or collide. Isolation policies can in-

tensify disturbance escalation; encryption-induced bandwidth pressure can amplify timing leakage; and row-buffer partitioning can conflict with attestation or monitoring frequency. As shown in Figure 4, the PIM attack surface spans from host interfaces and firmware orchestration to subarray-level fault phenomena, confirming that isolated countermeasures routinely overlook emergent interactions between layers. Figure 4 also identifies hardware trojans at the memory controller and logic layer as a supply-chain threat that the mechanisms surveyed in Sections 5.1–5.6 do not address; this constitutes an additional gap beyond those captured in the four categories below. Comparative analysis across the surveyed systems and Table 4 reveals four recurring systemic gaps, divided into architectural and methodological categories.

Architectural gaps:

- The absence of a cross-layer security plane capable of enforcing global policies across host, controller, and subarray levels;
- Static trust models that do not accommodate disaggregated or multi-accelerator PIM deployments, where device provenance and tenant boundaries shift at runtime.

Methodological gaps:

- Inconsistent evaluation methodologies across surveyed works that inhibit reproducibility and cross-paper comparison;
- Limited validation under large, heterogeneous, multi-tenant workloads representative of cloud-scale deployment.

While PIM security research has achieved substantial depth within individual threat domains, it lacks architectural convergence and coordinated design principles. This fragmentation complicates systematic reasoning about what must be protected, who the adversaries are, and where trust boundaries reside. Section 6 formalizes the adversary model that structures this analysis; Section 7 proposes the design framework.

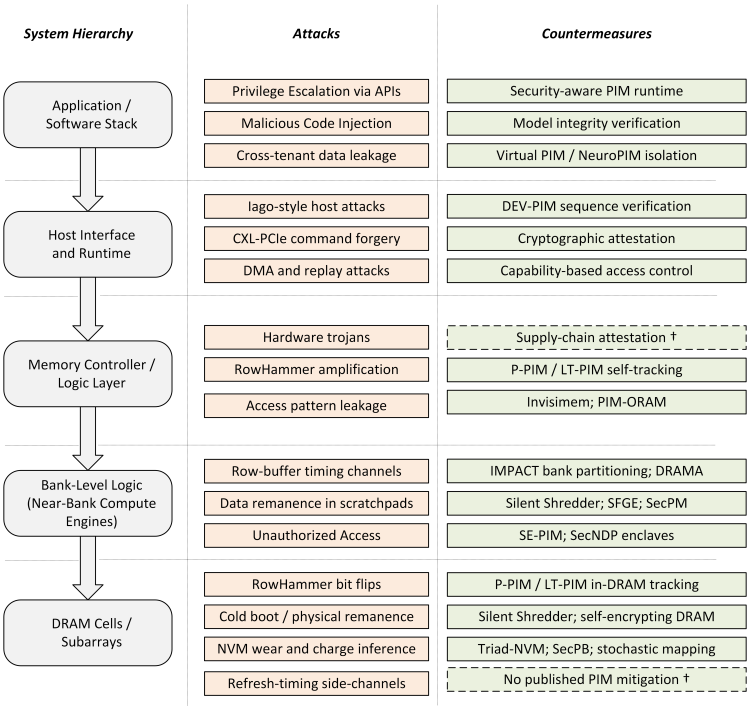


Figure 4. Threat attack surface of secure PIM architectures across the system hierarchy (**left**). Attacks (**center**) and countermeasures (**right**) are mapped to each hardware and software layer. Dashed-border countermeasure boxes (†) indicate open problems for which no PIM-specific mitigation has been published in the surveyed literature (2014–2025).

Table 4. Threats and Representative Defenses in PIM Systems.

Threat Class	Description in PIM Context	Representative Mitigations	Representative Works	PIM Platform	Limitations/Open Gaps
Host–PIM Orchestration	Compromised host OS or hypervisor can forge PIM descriptors or reorder dispatch to subvert execution (Iago-style). CXL/PCIe commands lack cryptographic integrity or replay protection.	Cryptographic attestation of instruction sequences; runtime verification against DRAM-resident metadata; lightweight hash and sequence-number schemes	DEV-PIM [47]; Checkoway & Shacham [42]; Ghinani et al. [71]	DRAM-PIM; CXL-PIM; FPGA-emulated PIM	Metadata overhead and cryptographic cost conflict with PIM throughput; no solution covers multi-tenant disaggregated CXL deployments
TEEs in PIM Cores	PIM cores need confidentiality and integrity when host and memory bus are untrusted. Static root-of-trust models conflict with disaggregated topologies; side-channels from shared refresh and sense-amplifier circuits are neglected in enclave validation.	MPC-based secret sharing across CPU TEE and PIM; in-DRAM ORAM; CXL-based freshness via smart memory; hardware enclave boundaries	SE-PIM [52]; SecNDP [53]; PIM-ORAM [60]; Toleo [54]	UPMEM; HMC 3D-stacked; CXL smart memory; NVM-PIM	Static trust boundaries fail under dynamic multi-tenant workloads; attestation does not scale to composable CXL arrays
Resource Isolation and Scheduling	Shared banks and row buffers enable cross-tenant interference. Static vault/DPU assignment cannot adapt to shifting workloads; boundaries are exploitable for RowHammer [41]. Dynamic scheduling reduces co-residency but not physical coupling.	Vault-level static assignment; priority-based multiplexing; software DPU task migration; bank-level access quotas	NeuroPIM [55]; Virtual PIM [56]; Siloz [41]	HMC 3D-stacked; UPMEM DRAM PIM	Software isolation insufficient against shared sense amplifiers and refresh circuitry; dynamic schemes narrow but do not close co-residency windows
Information Leakage and Side Channels	Row buffers, bank I/O, and NVM on-DIMM caches leak via timing, access-pattern, power, and wear-leveling observations. PIM DMA bypasses CPU caches, amplifying row-buffer timing channels.	Address encryption with uniform bus activity; in-DRAM ORAM; bank-level row-buffer partitioning; PIM instruction rate-limiting; NVM power-side-channel masking; NVM cache-set partitioning	InvisiMem [59]; IMPACT [36]; PIM-ORAM [60]; PME [62]; NVLeak [61]	DRAM-PIM (UPMEM, HMC); NVM-PIM (Optane); CXL-PIM	Full mitigation incurs high overhead; bank-level partitioning remains an open problem [36]; InvisiMem requires smart-memory hardware unavailable on commodity DDRx
RowHammer and Memory Reliability	PIM compute units generate row activations independently of the host; host-managed TRR cannot observe or anticipate these patterns. Both benign high-frequency kernels and malicious co-tenant hammering can exceed disturbance thresholds. Non-uniform patterns bypass TRR on all tested DDR4 devices [11].	In-DRAM XNOR integrity checks on high-activation rows; LUT-based periodic self-tracking; dual/triple-row activation detection; CXL-attached version storage for freshness	P-PIM [13]; LT-PIM [12]; Blacksmith [11]; Toleo [54]	Programmable DRAM; LUT-equipped DRAM PIM; CXL smart memory	Stealthy non-uniform hammering may evade periodic XNOR checks; P-PIM and LT-PIM target centralized controllers; no mitigation covers disaggregated or CXL-attached PIM
Data Remanence and Secure Erasure	NVM cells retain plaintext after power-off. Crash-recovery inconsistency exposes encrypted data. PIM scratchpads and in-flight buffers hold sensitive intermediates that host-managed zeroization cannot reach.	Counter-repurposing for zero-cost shredding; counter-mode encryption with crash-consistent persistence; selective Merkle Tree persistence; battery-backed buffers; just-in-time weight encryption	Silent Shredder [63]; SecPM [64]; Triad-NVM [66]; SecPB [67]; SFGE [65]	NVM-PIM (PCM, RRAM); NVM controllers; PIM cores (ML)	Centralized-controller designs do not extend to distributed PIM scratchpads; no mechanism identifies security-sensitive intermediates in distributed execution; simultaneous erasure across bitline buffers, in-flight state, and NVM layers is unsolved

6. Adversary Model for PIM Systems

The attack surface of a Processing-in-Memory (PIM) system reflects the heterogeneity of its execution pipeline and the erosion of traditional trust boundaries inherent to memory-side computation. Prior analyses address individual threat classes in isolation—SecNDP and PIM-Enclave focus on descriptor authentication [39,53], DEV-PIM and IMPACT characterize microarchitectural inference channels [36,47], and P-PIM and LT-PIM examine row-disturbance effects [12,13]—yet no prior work consolidates adversary capabilities, architectural boundaries, and success conditions into a unified structure suitable for systematic defense evaluation.

Four adversary classes operate within this threat environment. First, a malicious co-tenant interacts with the PIM runtime through sanctioned SDKs and command queues, issuing arbitrary kernels, crafting operand layouts, and modulating contention to infer or

influence a victim's execution. Second, a compromised host OS or hypervisor exerts far broader control, capable of remapping physical memory, forging PIM descriptors, altering dispatch order, or shaping scheduling metadata. Third, a malicious or compromised firmware entity such as corrupted controller microcode or an altered in-memory instruction interpreter, poses an even more potent threat by redefining the semantics of memory-side execution. Fourth, a limited physical adversary with transient access to memory channels or DIMM connectors is bounded as a passive observer: it may monitor channel signaling and recover remanent state, but it cannot inject signals, induce faults, or perform invasive chip modification [12,13]. None of these adversaries are assumed capable of breaking standard cryptography or subverting hardware roots of trust that are securely provisioned into the system.

Using the adversary capabilities defined in Table 5, the STRIDE mapping in Table 6 was constructed as follows. For each architectural boundary, each of the six STRIDE categories was evaluated: a category was assigned if a concrete attack mechanism exploiting that boundary appears in the surveyed literature or follows directly from the boundary's properties.

Table 5. Adversary Classes in the PIM Threat Environment.

Adversary Class	Capability	Attack Surface	Attack Mechanisms	STRIDE	Works
Malicious co-tenant	Low–medium	SDK/command queue; operand layout; row buffer	Kernel injection; contention modulation; timing inference; subarray interference	I, D	[12,13,36,47]
Compromised host OS/hypervisor	High	Host–PIM interface; PIM descriptors; dispatch scheduler	Physical memory remapping; descriptor forgery; dispatch reordering; scheduling metadata manipulation	S, T, E	[39,53]
Malicious/compromised firmware	Very high	Controller microcode; in-memory instruction interpreter	Semantic redefinition of memory-side execution; privilege escalation via microcode	T, E, S	[39,53]
Limited physical adversary	Medium	Memory channels; DIMM connectors; DRAM substrate	Passive channel observation; remanent state recovery. Following are excluded: active signal injection, fault induction, chip decapsulation [12,13]	I, T	[12,13,64,66]

STRIDE codes: S = Spoofing, T = Tampering, R = Repudiation, I = Information disclosure, D = Denial of service, E = Elevation of privilege.

Table 6. STRIDE Threat Mapping by Architectural Boundary (Section 6).

Architectural Boundary	STRIDE	Attack Examples	Adversary Class	Scope
Host–PIM interface	S, T, E, R	Forged descriptors; unauthorized kernel execution; command queue log erasure	Host OS/firmware	In scope
PIM logic (microarchitecture)	I, T	IMPACT-style covert channels; operand-dependent timing	Co-tenant	In scope
DRAM substrate (subarray)	T, I, D	RowHammer disturbance; row-buffer contention; activation-pattern leakage	Co-tenant; physical adversary	In scope
NVM persistent storage	I, R	Remanence recovery; rollback reconstruction	NVM-persistent attacker	In scope
Chip internals (decap/microprobe)	–	Invasive physical attacks	Out-of-scope adversary	Excluded
Cryptographic layer/provisioned roots of trust	–	Breaking standard cryptography; subverting securely provisioned hardware roots of trust	Out-of-scope adversary	Excluded

STRIDE codes as in Table 5. Rows marked Excluded define the outer boundary of the threat model; defenses for those vectors are outside the scope of this work.

Spoofing at the host–PIM interface arises because PIM descriptors carry no cryptographic origin binding in current implementations [44,53]; a compromised host OS can therefore forge valid-looking descriptor sequences. Tampering at the DRAM substrate arises from RowHammer's demonstrated ability to flip bits in physically adjacent rows without read permission on those rows [13,24]. Denial of service at the DRAM substrate arises from IMPACT's demonstrated 8.2–14.8 Mb/s covert channel, which requires sustained high-rate activations that consume row-buffer cycles available to co-resident tenants [36]. Repudiation at the NVM persistent storage layer arises because no surveyed NVM-PIM mechanism provides a tamper-evident audit log of scratchpad writes; rollback attacks can therefore erase evidence of unauthorized access [66]. Repudiation risk extends beyond

NVM persistence to the DRAM controller command queue: in current PIM architectures, command queues that schedule row activations, precharges, and PIM kernel dispatches maintain no tamper-evident log of issued commands. A compromised firmware entity or privileged attacker can issue unauthorized PIM operations and subsequently overwrite the command queue entries, eliminating evidence of the unauthorized execution. Implementing command-queue auditing within the DRAM controller would require appending a cryptographic hash or sequence number to each dispatched command and persisting these audit records in a write-protected region of the memory controller's SRAM, analogous to DEV-PIM's instruction-sequence verification [47] but extended to cover all memory-side command types rather than CPU-side instructions only. The overhead of such auditing, in terms of both command-queue depth reduction and verification latency, has not been evaluated in any published PIM work. Elevation of privilege at the host–PIM interface is the success condition of descriptor forgery (above); it is listed separately because its precondition (insufficient descriptor authentication) differs from the spoofing precondition (absent origin binding). Information disclosure appears at four boundaries because timing, spatial, and remanence channels each require a distinct countermeasure class; collapsing them into a single entry would obscure the defense design space.

These threats arise at multiple architectural boundaries, mapped to their corresponding STRIDE categories in Table 6. At the host–PIM interface, untrusted software can construct descriptor sequences or operand placements that coerce PIM logic into privilege escalation; SecNDP and PIM-Enclave enumerate these risks by demonstrating how insufficiently authenticated descriptors enable unauthorized execution [39,53]. Within the PIM logic itself, micro-architectural timing and operand-dependent behaviors, as characterized by DEV-PIM and IMPACT [36,47], create channels through which co-tenants may infer sensitive data. At the DRAM substrate, subarray-level interference, activation patterns, and row-buffer reuse amplify information leakage in ways not encountered in CPU-centric memory models. Row-disturbance failures identified in P-PIM and LT-PIM [12,13] demonstrate how manipulation of memory physics can corrupt nearby state or tamper with PIM-resident data. Persistent storage paths create additional attack vectors through persistent state: remanence and rollback risks documented in SecPM and TriadNVM [64,66] confirm how stale data may survive beyond its intended lifecycle, enabling cross-tenant recovery attacks in non-volatile memory (NVM) subsystems.

Trust in this environment resides in the secure microcode of the PIM controller, its embedded roots of trust, and the correctness of DRAM signaling at the physical level. The host software stack, including OS, hypervisor, libraries, and user runtimes, is considered untrusted. Co-tenants are likewise untrusted, even when logically isolated at the bank or subarray level by OS-enforced address-range partitioning. Such partitioning constrains memory addressing but does not prevent timing-based inference or row-disturbance across subarray boundaries, as demonstrated by IMPACT and P-PIM [13,36]. Fully invasive attacks requiring chip decapsulation or microprobing are excluded, as the primary deployment context addressed by this survey is cloud and data center PIM where an adversary is assumed to lack sustained unsupervised physical access to individual DIMMs; supply-chain threats operating prior to deployment are noted as an open gap in Section 5.7 but are outside the scope of the adversary model formalized here. Timing, contention, micro-architectural interference, firmware compromise, and remanence-based observations remain fully in scope.

Across adversary classes, success conditions include exfiltrating data or model parameters through timing or spatial inference, escalating from unprivileged PIM kernels into privileged control domains, silently corrupting or biasing computation within banks or subarrays, and denying service by exhausting PIM engines, row-buffer cycles, or controller

queues. These conditions structure a STRIDE threat analysis applied to the host–PIM interface, the PIM-DRAM micro-architecture, and inter-tenant sharing domains; the mapping of each boundary to its applicable STRIDE categories appears in Table 6. Spoofing and elevation of privilege manifest as forged descriptors or compromised microcode; tampering and information disclosure arise from subarray interference and operand-dependent execution; and denial of service emerges through orchestrated contention. Representative attack paths, including IMPACT-style covert channels, RowHammer-like disturbance escalation, and remanence-driven reconstruction, are captured as attack-tree sequences (Table 7), observed in existing prototypes.

The resulting threat model formalizes the adversarial environment in a manner suitable for evaluating existing defenses and motivating the secure-by-design PIM substrate proposed in Section 7.

Table 7. Representative Attack Paths in the PIM Threat Environment.

Attack Path	Adversary Class	Exploitation Sequence	Success Condition	Works
IMPACT-style covert channel	Malicious co-tenant	(1) Sender modulates operand-dependent PIM execution timing; (2) receiver samples row-buffer latency via shared subarray; (3) receiver decodes bit stream from latency variation	Data or key exfiltration across tenant boundary	[36,47]
RowHammer-like disturbance escalation	Malicious co-tenant; limited physical adversary	(1) Adversary issues repeated activations to aggressor rows adjacent to victim subarray; (2) accumulated charge disturbance flips bits in victim row; (3) flipped bits corrupt PIM-resident data or page-table entries	Integrity violation; potential privilege escalation	[12,13]
Descriptor forgery and privilege escalation	Compromised host OS/hypervisor	(1) Attacker constructs malformed PIM descriptor referencing privileged memory region; (2) insufficient descriptor authentication passes validation; (3) PIM kernel executes with escalated privilege	Unauthorized execution in privileged control domain	[39,53]
Remanence-driven cross-tenant reconstruction	NVM-persistent attacker	(1) Victim tenant deallocates NVM pages; (2) stale data persists beyond intended lifecycle due to absent sanitization; (3) subsequent tenant reads residual data from reallocated pages	Cross-tenant data recovery; model parameter exfiltration	[64,66]
Firmware-based semantic redefinition	Malicious/compromised firmware	(1) Adversary compromises PIM controller microcode; (2) altered microcode redefines memory-side execution semantics; (3) host-visible behavior diverges silently from specification	Silent computation corruption; covert persistent backdoor	[39,53]

Exploitation sequences are presented as ordered steps from initial adversary capability to success condition. Each path corresponds to an adversary class defined in Table 5 and a boundary entry in Table 6.

7. Conceptual Framework for Security-Centric PIM

PIM security research has reached a point where ad hoc or compartmentalized defenses demonstrate fundamental limitations that cannot be resolved through incremental refinement of existing CPU-centric models. Existing CPU-centric security models, based on monolithic enclaves, software-enforced access control, or top-down attestation, treat memory as a passive storage medium. The five mechanisms proposed here draw on primitives that appear individually in prior NDP security work; the contribution is their synthesis as a memory-resident compositional unit with explicit inter-mechanism dependencies. PIM introduces a structurally different condition: execution, communication, and data persistence coexist within the same physical domain. This section proposes a conceptual security framework that addresses this structural gap by locating trust enforcement within the memory substrate itself. No prototype of this framework has been implemented; however, a preliminary simulation characterizing DRAM-interface-level sensitivity is presented in Section 7.1. The contribution remains a structured design proposal intended to guide future experimental and formal validation. This positioning is consistent with prior architectural proposals in the PIM and near-data processing literature that established design foundations before implementation, including early near-data processing security models [53] and memory-side encryption frameworks [64]. The proposed framework is intentionally positioned as a security architecture specification rather than a circuit-level implementation.

It defines the enforcement locations, security invariants, and interaction requirements of five memory-native mechanisms. The preliminary simulation characterizes selected DRAM-interface-level sensitivity costs; circuit-level realization, multi-bank scheduling, and adversarial multi-tenant validation remain necessary steps before implementation-level performance or area claims can be made. Conventional enclave architectures, such as Intel SGX and ARM TrustZone, confine threats within CPU-controlled contexts and assume a single, central root of trust. Such models cannot scale to the distributed, high-bandwidth, and disaggregated topology of PIM, where thousands of compute units share banks, subarrays, and data buses. Rather than adapting these CPU-centric models, the proposed framework treats computation locality, timing, and access patterns as first-class elements of the security design. Specifically, it organizes five conceptual mechanisms—subarray-level isolation, temporal compartmentalization, distributed trust anchoring, data-oblivious execution, and adaptive policy orchestration—as complementary layers of a memory-resident defense model of the PIM threat environment. Table 8 positions each mechanism against prior PIM security proposals across five security properties: access-control granularity, data lifetime control, attestation topology, access-pattern confidentiality, and policy adaptability.

Table 8. Security Requirements for a Composable PIM Substrate, Derived from Gaps Identified in Section 5.

Req.	Requirement	Gap It Addresses (Section 5)	Framework Primitive (Section 8)
R1	Enforce tenant isolation at subarray granularity, independent of OS-level address-range partitioning	Physical-layer side channels persist under software-only isolation (Section 5.3)	Subarray-level isolation
R2	Bound data lifetime at the hardware layer; guarantee erasure of PIM-resident intermediates at end of scheduling epoch	NVM and DRAM scratchpad remanence unaddressed by existing defenses (Section 5.6)	Temporal compartmentalization (TTL-bounded scrubbing)
R3	Verify PIM execution integrity without trusting the host OS or hypervisor	Host-dependent attestation hierarchies broken by CXL disaggregation (Sections 5.1 and 5.2)	Distributed trust anchors
R4	Provide access-pattern confidentiality for PIM kernel execution	Row-buffer and memory-bus timing channels remain open after isolation (Section 5.4)	Data-oblivious execution pathways
R5	Support runtime adjustment of isolation and erasure policies across heterogeneous tenants	Static threat models cannot accommodate dynamic multi-tenant workload boundaries (Section 5.7)	Adaptive policy orchestration
R6	Compose with adjacent defense layers without undetermined aggregate overhead	No published end-to-end security budget analysis exists across any defense stack (Section 5.7)	(Open validation requirement—not addressed by any existing or proposed mechanism)

Requirements are derived from the four recurring systemic gaps in Table 4 and the adversary class success conditions in Table 5. R6 is listed without a framework primitive because no mechanism in the surveyed literature or in this proposal quantifies compositional overhead; it defines the primary open research problem identified in this work.

The first mechanism is subarray-level isolation. Existing PIM defenses, including SecNDP and PIM-Enclave, enforce access control at page or bank granularity. The proposed framework instead locates enforcement at wordlines or bitline groups, the smallest physical granularity of DRAM. This finer boundary is motivated by the observation that adversarial computation can exploit spatial interference or sense-amplifier coupling within a single bank, a threat that coarser isolation boundaries do not contain. Conceptually, lightweight access-control metadata embedded within the local row decoder would associate each compute request with an authenticated context, binding privilege domains to their physical locality. Embedding digital checking circuits in the peripheral control logic of high-density DRAM raises physical feasibility concerns. DRAM row decoders operate within stringent physical layout constraints: the wordline pitch in modern DDR4/DDR5 nodes is 30–40 nm, and the row-activation timing window (t_{RCD}) is approximately 13–14 ns. Inserting metadata-check logic into this critical path risks both throughput degradation from increased t_{RCD} and signal integrity deterioration from additional routing in the sense-amplifier peripheral region. The proposed metadata is therefore not intended to be embedded in the row decoder's critical timing path. Instead, it would be stored in a dedicated SRAM tag array adjacent to the row decoder (analogous to the tag arrays used in

CACTI-modeled cache structures), with the metadata check performed in parallel with the row-activation precharge phase rather than serialized with the wordline assertion. Under this design, the metadata check consumes the precharge interval ($t_{RP} \approx 13$ ns) rather than adding to t_{RCD}, resulting in zero additional latency for row-hit accesses and one t_{RP} penalty only for row-conflict accesses that require a metadata context switch. The area overhead of the tag array is estimated at 4–16 bits per row $\times$ the number of rows per subarray; for a 1024-row subarray with 8-bit tags, this yields 1 KB of SRAM per subarray, which is less than 0.1% of the subarray's data capacity.

The proposed metadata store is modest in capacity: 1 KB per subarray represents approximately 0.1% of the data capacity in an illustrative 1024-row, 1 KB-per-row subarray organization. Existing RowHammer mitigations demonstrate the architectural use of activation-history metadata and targeted refresh logic at bank granularity, although the detailed circuit placement and implementation of commercial TRR mechanisms are proprietary [11,70]. Accordingly, we use TRR only as evidence that production DRAM can support bank-level metadata tracking; it does not establish that our tag-array design has the same physical implementation. The framework assumes a peripheral tag store and authorization path that operates in parallel with row-state transition handling. In DDR4-2400, t_{RP} is approximately 13 ns; the preliminary evaluation therefore sweeps an added activation-path delay of 2–10 ns to cover a range from lightweight tag comparison through a conservative policy-check abstraction. A realizable design must ensure that metadata lookup, context comparison, and gating complete within the controller's scheduled row-transition interval, or otherwise account for the residual delay explicitly. The present work does not claim timing closure; it identifies the t_{RP} interval as the design-target window and bounds the overhead sensitivity through simulation. Tag updates are assumed to occur only at security-context reassignment (e.g., tenant migration), not on ordinary data accesses. Their scheduling and atomicity must be coordinated with bank quiescence, refresh scheduling, and controller state transitions; evaluating those interactions requires a modified DRAM-controller model and circuit/layout data unavailable in the present study. Likewise, routing-density, coupling, and process-compatibility effects require a proprietary DRAM PDK. The discussion therefore establishes architectural plausibility and identifies design constraints, rather than demonstrating final physical implementation feasibility.

Preliminary gem5 simulation results quantifying the timing overhead of this metadata check are presented below. Unlike static firewalling, this isolation is designed to be dynamically scalable, with context boundaries shifting as workloads migrate or memory is reconfigured under all runtime configurations. Evaluating this mechanism would require DRAMSim3 coupled with a RISC-V-based PIM ISA to emulate metadata propagation latency and verify context isolation under varied bank utilization scenarios [58]; such evaluation is identified as future work. The applicability of subarray-level isolation to Logic-in-Memory designs warrants explicit discussion. LiM architectures such as Ambit operate through charge-sharing mechanisms at the bitline level, with no programmable logic capable of hosting metadata or context tags. Embedding access-control metadata within a LiM row decoder therefore requires adding digital logic to an analog compute substrate, a modification that increases area and may degrade the charge-sharing margins on which LiM computation depends. The honest implication is that the subarray-level isolation mechanism as described applies most directly to NDP-class PIM, where a programmable logic layer exists to host the metadata. For LiM, the appropriate security primitive is likely coarser: bank-level address-range enforcement enforced by the memory controller rather than within the array, accepting a weaker isolation guarantee in exchange for preserving LiM's efficiency. This distinction is reflected in Table 8, where R1 is scoped to designs with programmable logic layers.

The second mechanism is temporal compartmentalization. Persistence mechanisms such as SecPM and TriadNVM address durability and consistency but do not control how long sensitive state remains accessible after a computation completes. The proposed framework addresses this by embedding time-to-live (TTL) metadata and monotonic counters within memory controllers, such that sensitive state or cached operands expire after a bounded interval and trigger secure overwrite or obfuscation routines. This design targets rollback, remanence, and post-crash forensics threats that persistence-focused prior work leaves unaddressed. The resulting construct is referred to here as a *secure execution cell*: a logical unit that autonomously manages its own data retention lifecycle within a single bank context. This differs from a PIM-Enclave or standard TEE in a specific and intentional way. Existing enclave designs, including SE-PIM and PIM-Enclave, establish confidentiality and integrity boundaries for computation, but do not bound how long data persists after the computation ends. Existing enclave designs bound who can access data. A secure execution cell bounds how long the data exists; the TTL counter triggers scrubbing regardless of access state, closing the remanence window that enclave-style access control leaves open. The differentiating primitive is therefore TTL-bounded scrubbing, not access control, which existing enclave designs already provide. Overhead is expected to be dominated by timer synchronization; simulation-level validation is required before any performance claim can be made.

For TTL counters to function accurately across disaggregated memory banks, a consistent time reference must be maintained across independently clocked domains. In HBM2e stacks, each bank operates within the memory clock domain (typically 1.2–1.6 GHz), while the host controller and CXL interconnect operate on separate clock domains. The proposed TTL counters would be clocked from the memory-side clock domain, using the bank's local row-activation clock as the time base. Cross-bank TTL consistency does not require strict clock synchronization; instead, each bank's TTL counter operates on its local clock, and the policy co-processor (mechanism five) reconciles TTL expiration events across banks using a loosely synchronized epoch protocol, analogous to the epoch-based reclamation used in concurrent data structures. The maximum allowable clock skew between banks determines the worst-case TTL enforcement granularity: at 1.6 GHz memory clock with $\pm 1\%$ clock skew across banks, the TTL enforcement window has a worst-case imprecision of ± 10 ns per microsecond of TTL duration, which is well within the timing margins of the scrubbing operations that TTL expiration triggers. For CXL-attached deployments where memory devices have fully independent clock domains, the host-side CXL controller would broadcast periodic synchronization beacons; the latency cost of these beacons is identified as a parameter requiring simulation-level characterization. Critically, TTL counter integrity depends on the distributed trust anchors described in the following mechanism: without anchor attestation, an adversary could forge or reset TTL counters, nullifying temporal confinement. This dependency is the primary inter-mechanism coupling in the proposed design.

The third mechanism replaces centralized enclaves with distributed trust anchors instantiated across memory banks. In prior work such as SE-PIM and PIM-Enclave, attestation flows through a static enclave hierarchy. The proposed model instead assigns each bank anchor the role of local verifier, responsible for attesting to the authenticity of PIM cores, managing ephemeral cryptographic keys, and validating operand integrity. The design specifies two attestation directions: horizontal chains across banks to prevent cross-bank impersonation, and vertical chains across hardware and software layers to authenticate orchestration commands. Horizontal attestation means that each anchor cryptographically verifies the identity of adjacent bank anchors before accepting cross-bank operands. Vertical attestation means that each anchor validates the command provenance of the orchestration

layer above it before executing policy updates. These anchors could be implemented in the logic layer of HBM2e or FPGA-based Near-Data Processing prototypes, with attestation verified through lightweight cryptographic checksums embedded in command queues. This distributed structure is motivated by PIM's physical disaggregation: a centralized root of trust is structurally incompatible with a topology in which computation and storage are distributed across hundreds of independent banks, each with co-located compute and storage resources.

A critical open question for this mechanism is how bank anchors acquire their initial identities in a vendor-disaggregated deployment, where the memory device and host controller may originate from different manufacturers. The proposed approach follows the manufacturer-provisioned endorsement model established in Sanctum [72], in which the processor manufacturer signs a hardware root key at fabrication time, enabling remote attestation without host involvement. Adapted to the memory substrate, each bank anchor's asymmetric key pair is generated and signed by the memory manufacturer during wafer-level test, with the endorsement certificate stored in one-time-programmable fuses in the logic layer. In a CXL-attached deployment, the host controller requests the anchor's endorsement certificate and validates it against the manufacturer's public root certificate, a process that requires no trusted host, only a trusted manufacturer PKI. Sanctum demonstrates this bootstrapping model on an open RISC-V implementation [72], the same ISA class targeted for NDP-class PIM logic layers, making the provisioning model directly portable. Specifying the certificate format, revocation mechanism, and chain-of-trust policy for PIM-specific deployments is identified as a concrete standardization target within the second research trajectory of Section 8.

When a bank anchor's key is compromised or a memory module is replaced in a disaggregated topology, the revocation mechanism must propagate the invalidation to all peer anchors that have established horizontal attestation chains with the compromised anchor. The proposed approach uses a certificate revocation list (CRL) distributed through the CXL controller's management channel: upon detecting a compromised anchor (via failed attestation challenge or external revocation signal from the manufacturer PKI), the CXL controller broadcasts a signed CRL update to all bank anchors in the affected CXL memory pool. Each anchor maintains a local CRL cache in its OTP-adjacent SRAM and rejects cross-bank operands from any anchor whose certificate appears on the CRL. During dynamic task migration, when a PIM workload migrates from one bank to another, the destination bank's anchor must (i) verify that the source bank's anchor is not revoked, (ii) establish a fresh ephemeral session key with the source anchor for secure transfer of the workload's cryptographic context, and (iii) re-attest the migrated workload's integrity before permitting execution. This three-step migration handshake introduces latency overhead proportional to the cryptographic operation cost (estimated at 100–500 ns for lightweight symmetric key exchange on the logic-layer process node), which must be amortized across the migration interval to remain within acceptable bounds. The precise latency and its interaction with PIM scheduling remain open empirical questions.

The fourth mechanism addresses data-oblivious execution. Full oblivious RAM (ORAM) is well established as impractical for bandwidth-heavy workloads due to its logarithmic bandwidth overhead [60,73]. The proposed framework instead identifies three candidate techniques, deterministic access scheduling, operand shuffling buffers, and oblivious loop unrolling, as potential intermediate approaches that reduce access-pattern leakage without incurring full ORAM cost. These techniques are proposed as design candidates, not validated mechanisms. The design target for each candidate technique is to reduce the mutual information between the observable memory access sequence and the sensitive operand value, subject to available memory bandwidth constraints. More precisely, let X

denote the sensitive operand value and Y the observable memory access sequence (row addresses and timing). The mutual information $I(X;Y)$ quantifies the information an adversary gains about X by observing Y. For an unprotected PIM kernel, $I(X;Y)$ is bounded above by the entropy $H(X)$ of the operand space. Each candidate technique targets a specific reduction mechanism: deterministic access scheduling forces Y to be independent of X (achieving $I(X;Y) = 0$ for the scheduled access component), operand shuffling introduces noise into the $X \to Y$ mapping (reducing $I(X;Y)$ by the entropy of the shuffling permutation), and oblivious loop unrolling eliminates data-dependent branch patterns (reducing the conditional entropy $H(Y|X)$ contribution from control flow). Quantifying $I(X;Y)$ for each technique under realistic PIM workloads requires collecting bank-conflict timing traces from cycle-accurate simulation and applying a k-nearest-neighbor mutual information estimator [74], a methodology established for cache side-channel analysis. This quantification is identified as future work; we note that the absence of formal mutual-information bounds is a limitation shared with all surveyed PIM defense mechanisms, none of which provide information-theoretic security guarantees.

Oblivious loop unrolling, which eliminates data-dependent branch behavior by executing all loop iterations regardless of early-exit conditions, can cause severe instruction cache (I-cache) thrashing in NDP cores. NDP-class PIM cores typically have small I-caches (e.g., UPMEM DPUs have a 24 KB instruction memory [58]); aggressively unrolled loops expand the instruction footprint beyond the I-cache capacity, converting I-cache hits into misses and degrading throughput. The performance trade-off is quantifiable: for an unrolling depth of d iterations with instruction body size b bytes, the unrolled footprint is $d \times b$ bytes. When $d \times b$ exceeds the I-cache capacity C, every unrolled iteration triggers an I-cache miss, with a miss penalty of $\sim$10–20 cycles on in-order NDP cores. The optimal unrolling depth limit is therefore $d^* = \lfloor C/b \rfloor$, beyond which the I-cache thrashing penalty exceeds the security benefit. For UPMEM DPUs with 24 KB instruction memory and a typical PIM kernel body of 64–256 bytes, this yields an optimal unrolling depth of 96–384 iterations. Beyond this limit, alternative obfuscation techniques such as operand shuffling buffers should be preferred. This trade-off must be evaluated per-platform, as I-cache sizes vary across NDP implementations. This mechanism couples with the distributed trust anchors of the third mechanism: anchor attestation extends to execution behavior, binding security guarantees to both the identity of the executing core and the obliviousness of its access schedule.

The fifth mechanism is a policy co-processor embedded within the memory controller. This unit is designed to interpret runtime directives and translate them into configuration updates for subarray access tags, TTL parameters, and oblivious scheduling modes. Conceptually, the co-processor receives three input signal classes: workload context descriptors from the host memory controller, tenant isolation directives from the orchestration layer, and attestation status flags from the distributed trust anchors. It produces two output classes: per-subarray configuration register updates and anchor reconfiguration commands. The motivation for this component is the heterogeneity of PIM deployment contexts: AI inference workloads have different isolation requirements than transactional memory operations, and a static security configuration cannot serve both without incurring unnecessary overhead in one case or insufficient protection in the other. Because the co-processor can issue reconfiguration commands to trust anchors, it must itself be treated as a trusted component; how the co-processor's own trustworthiness is established, and whether it requires a separate bootstrap attestation step, is an open design question identified for future specification. The interface specification, latency budget, and interaction protocol with the attestation layer are scoped as future design work. On attestation failure, the co-processor halts configuration updates to affected subarrays until attestation is re-established; the precise failure semantics require formal specification.

Figure 5 illustrates the conceptual relationship among these five mechanisms. The figure is architectural rather than implementational; it represents design intent and does not reflect measured or simulated behavior. As shown, the Host OS/Hypervisor layer issues policy directives and context metadata downward to the Distributed Root-of-Trust layer, whose bank-local trust anchors perform local attestation and integrity provenance synchronization with the PIM Execution layer below. Each PIM core incorporates a capability verifier, and an oblivious scheduler mediates access-pattern scheduling across cores. At the DRAM bank layer, each bank is augmented with an access-control tag block, privilege-tagged row decoders, TTL timers, erase controllers, and integrity checkers that enforce subarray-level isolation and temporal compartmentalization. Dynamic subarray trust zones, shown for two tenants and one isolated fault region, illustrate how context boundaries shift at runtime. Attestation feedback and integrity reports flow upward from the trust anchor layer to the host workload scheduler, closing the vertical attestation chain. The primary structural distinction from prior work is that the framework locates the root of trust within the memory array itself, requiring host-side software to obtain attestation from memory rather than imposing it upon memory. Whether this structural inversion produces measurable security or performance advantages over existing approaches is an open empirical question that the framework is intended to motivate.

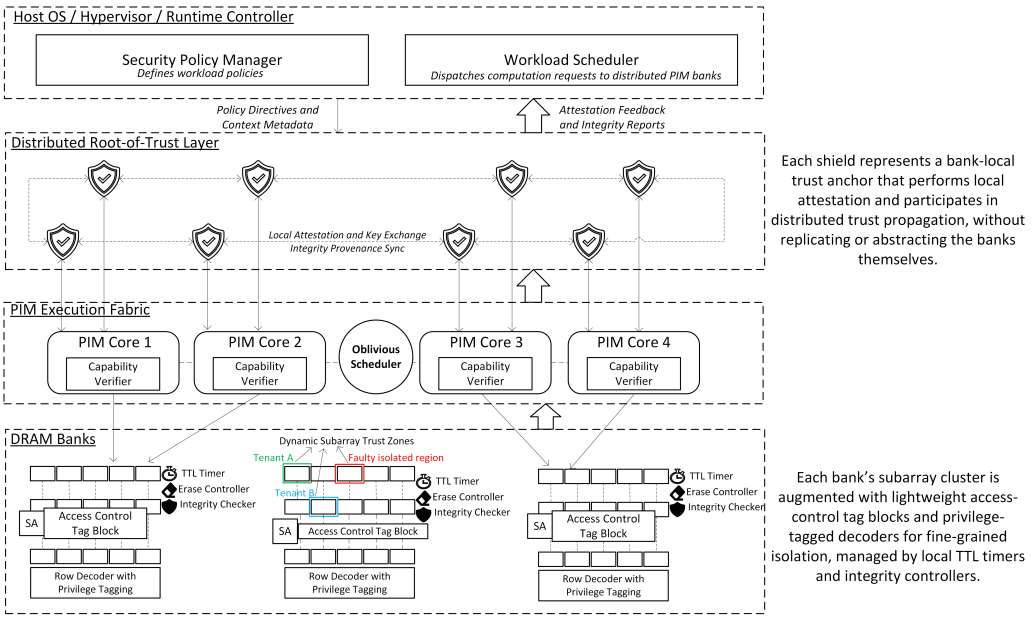


Figure 5. Proposed conceptual architecture of the memory-native PIM security framework, illustrating the intended interaction among subarray-level isolation, temporal compartmentalization, distributed trust anchors, the oblivious scheduler, and the policy co-processor across four architectural layers. Shield icons represent bank-local trust anchors performing local attestation and integrity provenance synchronization. Dynamic subarray trust zones show per-tenant context boundaries and fault isolation at runtime. No implementation or simulation of this architecture has been conducted; the diagram represents design intent only.

The five mechanisms described above are designed primarily for NDP-class architectures with programmable digital controllers and, with caveats, for DRAM-based LiM designs. For purely analog compute-in-memory substrates, the proposed controller-centric mechanisms do not secure leakage originating within analog matrix–vector multiplication itself. Relevant leakage mechanisms include: (i) data-dependent current and power signatures during crossbar computation, which can reveal operand values through supply-current variation; (ii) device-state variation and drift in ReRAM/PCM cells, which causes stored weights to shift over time in a device-history-dependent manner detectable through

repeated measurements; and (iii) information leakage through analog-to-digital conversion behavior, where quantization patterns at the crossbar output can correlate with input data. Mitigations such as stochastic bit masking [15], noise shaping or injection, calibration and refresh procedures, and differential sensing must be implemented at the device or mixed-signal-periphery level and can incur energy, area, accuracy, or throughput costs whose magnitudes are implementation-dependent. These device-level mechanisms are complementary to, rather than replaceable by, digital access-control metadata.

Many production and near-production CiM designs are hybrid, combining analog crossbar arrays for matrix–vector multiplication with digital controllers for non-linear activation, data routing, and memory management [29]. For such hybrid architectures, the framework's applicability boundary falls at the analog-to-digital converter (ADC/DAC) interface: the five proposed mechanisms apply to the digital controller side (scheduling, access control, attestation, policy orchestration), while crossbar-internal leakage requires device- and mixed-signal-level protection. The ADC/DAC boundary is therefore a useful architectural point at which to define responsibility, but not a complete security boundary; a generally accepted composable interface that bridges digital-layer and device-layer security across this divide remains an open research problem. Table 9 summarizes the applicability of each framework mechanism across PIM substrate types.

Table 9. Framework mechanism applicability across PIM substrate types. Applicability reflects the enforcement model of each mechanism relative to the architectural features of each substrate class.

Mechanism	NDP (Digital Controller)	LiM (DRAM Charge-Sharing)	Analog CiM (ReRAM/PCM Crossbar)
Subarray-level isolation	Directly applicable; metadata tags in peripheral SRAM	Applicable at controller granularity; subarray-level enforcement requires additional DRAM-periphery logic and validation of array-timing and charge-sharing margins	Digital-periphery support plus device/mixed-signal protection required; access-control metadata alone cannot secure crossbar-internal leakage
Temporal compartment.	Directly applicable; counters in controller SRAM	Applicable at controller granularity; DRAM controller manages TTL, although realization below controller granularity would require additional DRAM-periphery support	Digital-periphery management plus device-aware retention/drift controls required
Distributed trust anchors	Directly applicable; per-bank verifiers in logic layer	Applicable at controller granularity; attestation possible at controller but not at subarray level	Digital periphery can host attestation logic; it cannot attest to or suppress analog-array leakage without additional device/mixed-signal support
Data-oblivious execution	Directly applicable; scheduling controlled by PIM controller	Applicable at controller granularity for scheduled operations; insufficient for charge-sharing computation itself	Digital-controller side only; scheduling may reduce externally visible traffic patterns, but cannot eliminate leakage from crossbar currents
Adaptive policy orch.	Directly applicable; co-processor in logic layer	Applicable at controller granularity; policy logic at memory controller	Digital-controller side only; policy can govern digital periphery but not analog array behavior
Required approach	This framework	This framework at controller granularity + bank-level address-range enforcement	Device-level countermeasures (masking, noise injection, differential sensing) + this framework for digital periphery

Validating this framework requires three methodological tracks. Functional evaluation using DRAMSim3 integrated with a RISC-V PIM ISA would quantify isolation latency, trust propagation overhead, and TTL enforcement cost under representative workloads [58]. Formal verification using symbolic model checkers such as CoSA or SymbiYosys would verify isolation invariance and TTL erasure completeness as defined above. Scalability analysis on FPGA or HBM2e-based prototypes would measure context synchronization latency and policy switching overhead, with attestation targeting the ≈35 ns HBM2e row-activation window [9]. Each track addresses a distinct class of claim and none substitutes for the others.

7.1. Preliminary Simulation Methodology

To provide an initial feasibility assessment of three key framework mechanisms, we use the gem5 simulator [75] to evaluate security overhead at the DRAM interface level. The simulation runs in syscall-emulation mode with a TimingSimpleCPU at 500 MHz

and a DDR4-2400 single-channel memory model. The configuration eliminates the cache hierarchy entirely: both instruction and data ports connect directly to the memory bus, ensuring that every load and store exercises the DRAM timing path where the proposed security mechanisms operate. This approximates the direct-DRAM-access behavior of NDP architectures, in which processing elements sit adjacent to DRAM banks and access memory without intervening caches, while remaining within stock gem5. Full NDP microarchitecture simulation, including bank-local scratchpad operation, host-to-NDP offloading concurrency, PIM-specific ISA extensions, and multi-bank parallel execution, is identified as future work requiring a custom simulator or gem5 fork. We parameterize three security-mechanism proxies independently: (i) subarray-level metadata checks, modeled as additional tRCD latency of 0, 2, 5, and 10 ns per row activation; (ii) data-oblivious padding, modeled as additional random memory accesses at 0%, 10%, 25%, 50%, and 100% of the original access count; and (iii) TTL counter maintenance, modeled as periodic DRAM writes at frequencies of 1/1000, 1/100, and 1/10 of the primary access rate. Three micro-benchmarks exercise distinct memory-access patterns: sequential scan (64 MB streaming), random access (uniform random over 64 MB), and strided access (4096-byte stride). Because no cache filters memory traffic, the measured overheads represent upper-bound estimates of the cost each mechanism imposes when every access reaches DRAM. We measure throughput degradation relative to the corresponding zero-overhead baseline for each benchmark. This setup constitutes a preliminary proof-of-concept evaluation at the DRAM interface level; it does not represent full system-level validation, which would require a distributed multi-channel PIM simulator with bank-local scratchpads, concurrent host-NDP traffic, and hardware-enforced isolation logic.

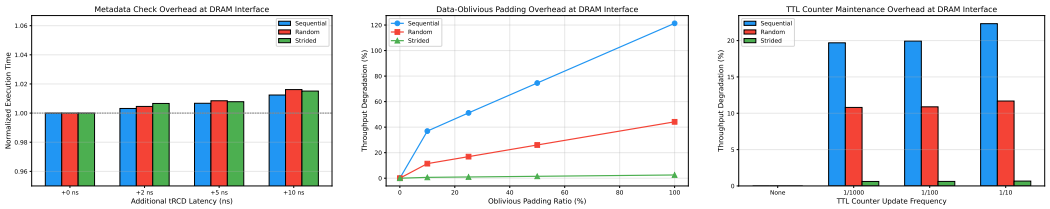
Table 10 summarizes the results across all three mechanisms and Figure 6 visualizes the overhead trends. Because the cache hierarchy is eliminated, all memory traffic, including instruction fetches, reaches DRAM, producing uniformly high baseline row-buffer hit rates (≈92–93%) across all three benchmarks. This uniformity arises because the sequential instruction stream dominates the overall access pattern regardless of data-access locality. Metadata-check overhead (Figure 6a) remains modest across all configurations: even at +10 ns tRCD, degradation stays within 1.2–1.6% because the additional latency is paid only on the ≈7% of accesses that are row-buffer misses. The uniformity across benchmarks confirms that subarray-level isolation tags are inexpensive when amortized over the dominant instruction-fetch traffic. Data-oblivious padding (Figure 6b) is the most expensive mechanism: the injected dummy accesses disrupt row-buffer locality, producing up to 121.3% degradation (≈2.2× slowdown) for sequential access and 44.2% for random access at 100% padding. Both are higher than they would be with caches, as expected: without cache filtering, every dummy access incurs full DRAM latency. Overhead scales near-linearly with padding ratio, providing a predictable cost model for deployment tuning. TTL counter maintenance (Figure 6c) imposes 19.7–22.3% overhead for sequential access and 10.8–11.7% for random access, with weak dependence on write frequency. This confirms that the cost is dominated by per-write row-buffer disruption rather than aggregate write bandwidth: even infrequent TTL writes force row closures that penalize subsequent accesses.

A critical question for any composable defense stack is whether overheads interact superlinearly when multiple mechanisms are deployed simultaneously. To evaluate this, we run composition experiments that combine all three mechanisms at three intensity levels: light (+2 ns tRCD, 10% padding, 1/1000 TTL), moderate (+5 ns tRCD, 25% padding, 1/100 TTL), and heavy (+10 ns tRCD, 50% padding, 1/10 TTL). For each configuration, we compute the predicted overhead as the linear sum of the three individual-mechanism overheads measured in Table 10 and compare it against the measured overhead of the combined deployment. The ratio of measured to predicted overhead yields an interaction

factor: values near 1.0 indicate linear (additive) composition, while values above 1.0 indicate superlinear interaction.

Table 10. Preliminary gem5 simulation results: throughput degradation (%) under parameterized security-mechanism overhead, measured on a cacheless DDR4-2400 single-channel configuration (500 MHz TimingSimpleCPU, no L1/L2 caches). All accesses reach DRAM directly, providing upper-bound overhead estimates.

Mechanism	Configuration	Sequential	Random	Strided
Metadata check	+2 ns tRCD	0.3%	0.5%	0.7%
Metadata check	+5 ns tRCD	0.7%	0.8%	0.8%
Metadata check	+10 ns tRCD	1.2%	1.6%	1.5%
Oblivious padding	10% padding	36.9%	11.4%	0.7%
Oblivious padding	25% padding	51.2%	16.9%	1.0%
Oblivious padding	50% padding	74.6%	26.0%	1.5%
Oblivious padding	100% padding	121.3%	44.2%	2.5%
TTL maintenance	1/1000	19.7%	10.8%	0.6%
TTL maintenance	1/100	19.9%	10.9%	0.6%
TTL maintenance	1/10	22.3%	11.7%	0.7%



(**a**) (**b**) (**c**)

Figure 6. Throughput overhead of three parameterized security mechanisms measured via gem5 simulation on a cacheless DDR4-2400 configuration. (**a**) Metadata check: normalized execution time under increasing tRCD latency; random access is most sensitive due to its low row-buffer hit rate. (**b**) Data-oblivious padding: sequential access degrades near-linearly as dummy accesses destroy spatial locality. (**c**) TTL counter maintenance: overhead is dominated by per-write row-buffer disruption rather than update frequency.

Table 11 presents the composition results. Across all nine benchmark–configuration combinations, interaction factors range from 1.00 to 1.02, confirming that the three mechanisms compose nearly linearly at the DRAM interface. The slight superlinear effect observed in the heavy-sequential configuration (1.02) is consistent with increased row-buffer contention when metadata latency, dummy padding accesses, and TTL writes all compete for the same bank: each mechanism's additional accesses marginally increase the probability that the next mechanism's access encounters a closed row. That this effect remains at 2% even under heavy combined load is a favorable result for the framework's composability thesis, as visualized in Figure 7. The near-linear composition supports the design assumption in Section 7 that security-mechanism costs can be budgeted independently and summed for system-level planning, at least at the DRAM-interface level evaluated here. The single-bank evaluation isolates per-request costs and does not capture shared-channel contention. In a multi-bank configuration, metadata checks are modeled as bank-local because they do not introduce additional external data transfers in the proposed design; their system-level effect is therefore primarily additional activation-path latency. In contrast, oblivious padding and TTL maintenance generate additional memory requests that consume shared command-bus, data-bus, and queue capacity. If baseline channel utilization is U and padding increases memory requests by fraction p, the first-order

offered-load estimate is $U(1+p)$. Workloads already operating near channel saturation will consequently experience nonlinear queueing growth at lower padding ratios than lightly utilized workloads. This effect is amplified by DRAM timing constraints, activation windows (t_{FAW}), bank-group restrictions, read/write turnaround penalties, and periodic refresh activity, each of which reduces the effective scheduling headroom available for security traffic.

The framework therefore treats adaptive policy orchestration (mechanism five) as a necessary control mechanism. A practical controller can monitor queue occupancy, row-buffer conflict rate, and per-bank utilization, signals commonly maintained by modern memory controllers, including queue occupancy, bank state, and request-stream locality indicators, and reduce padding intensity or defer TTL maintenance when contention exceeds a configured threshold. The stability behavior of such a controller and its security–performance trade-off curve require full-system multi-bank evaluation, which is identified as the first phase of the validation roadmap below.

Table 11. Composition analysis: measured throughput degradation when all three security mechanisms are deployed simultaneously, compared against the linear sum of individual overheads. An interaction factor <1 indicates sublinear (better than additive) composition; >1 indicates superlinear interaction.

Config	Benchmark	Predicted	Measured	Interaction
Light (+2 ns, 10%, 1/1000)	Sequential	56.9%	57.3%	1.01
	Random	22.7%	22.6%	1.00
	Strided	1.9%	1.9%	1.00
Moderate (+5 ns, 25%, 1/100)	Sequential	71.8%	72.5%	1.01
	Random	28.6%	28.8%	1.01
	Strided	2.4%	2.4%	1.00
Heavy (+10 ns, 50%, 1/10)	Sequential	98.1%	99.6%	1.02
	Random	39.3%	39.8%	1.01
	Strided	3.7%	3.7%	1.01

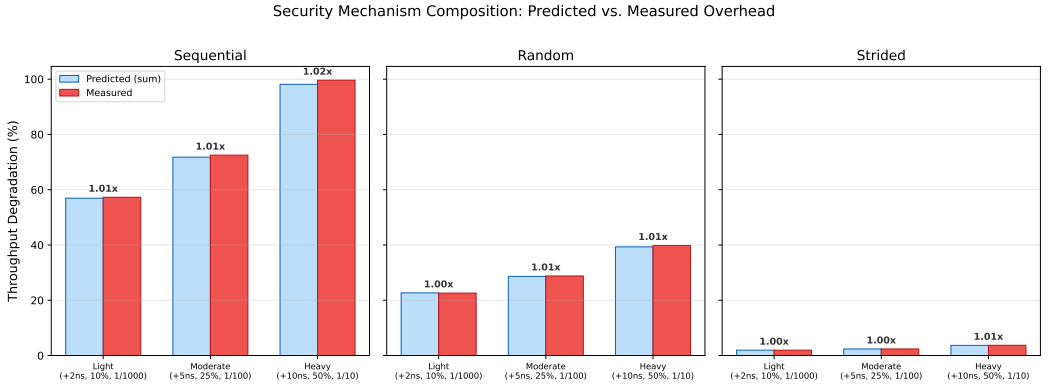


Figure 7. Composition overhead: measured vs. predicted throughput degradation when all three security mechanisms are deployed simultaneously. Each bar group pairs the predicted overhead (sum of individual mechanism overheads, blue) with the measured overhead under simultaneous deployment (red). The ratio annotations (1.00×–1.02×) indicate that measured overhead closely tracks the additive prediction, with slight superlinear deviation at higher overhead due to modest row-buffer contention effects.

Future compiler-controller co-design could further reduce bandwidth pressure: grouping accesses within security domains to preserve row-buffer locality, and scheduling padding during latency-tolerant periods rather than as independent additive operations. The present study does not model such compiler transformations; defining the compiler–

hardware interface for security-aware PIM memory scheduling is identified as a concrete target for future work.

Within the modeled single-channel DRAM-interface configurations, metadata-check overhead remains modest for locality-friendly workloads. Oblivious padding imposes the primary performance cost and is the natural target for optimization (e.g., adaptive padding rates tied to observed contention). Within the modeled update frequencies, TTL-maintenance overhead remains bounded. Across all configurations, the three mechanisms compose near-linearly when deployed together.

7.2. Validation Roadmap

Advancing from this proof-of-concept to system-level validation requires three evaluation phases, each targeting a distinct class of concern. Phase 1 (multi-bank composition): extend the gem5 configuration to a multi-bank DDR4-2400 model with per-bank security parameterization, measuring shared command/data-path contention, bank-group restrictions, refresh interactions, and the stability of adaptive policy control under different utilization levels. Phase 2 (multi-tenant interference): add two independent processing cores sharing the memory subsystem with distinct security contexts, evaluating isolation enforcement, timing leakage, and performance under adversarial co-tenant access patterns modeled on demonstrated DRAM/PIM-style covert-channel access patterns [36]. Phase 3 (NDP microarchitecture): integrate a RISC-V-based PIM processing element with bank-local scratchpad into a custom gem5 module, enabling evaluation of the full security framework including trust-anchor attestation latency and policy orchestration overhead. Together, these phases define the evidence needed before translating the present framework into implementation-level area, timing, and full-system performance claims.

7.3. Implementation Limitations

The five mechanisms proposed in this framework are specified at the architectural level. They identify security functions, intended enforcement locations, and interaction requirements, but they do not provide an RTL implementation, physical layout, complete key-management protocol, or formal leakage proof. The following limitations highlight the additional areas of work that must be addressed before practical implementation and deployment claims can be substantiated:

- *OS and hypervisor interaction*: The framework locates trust enforcement within the memory substrate and treats the host software stack as untrusted (Section 6). A deployable design nevertheless requires interfaces through which the host communicates context assignments, memory allocation, tenant migration, DMA offloading, and failure recovery to the memory-side enforcement logic. Because the adversary model grants a compromised host the ability to remap memory, forge descriptors, and manipulate scheduling metadata, these interfaces must ensure that the host cannot unilaterally establish, modify, or revoke security contexts without memory-side verification. Specifying such interfaces and their trust semantics is an open design challenge.
- *Trust-anchor provisioning and TTL synchronization*: Distributed trust anchors require a concrete lifecycle-management protocol for device enrollment, credential provisioning, attestation, key rotation, revocation, and recovery after component replacement or compromise. Because the adversary model includes compromised host software and controller firmware, these operations must be protected against counterfeit enrollment, suppressed revocation, replayed attestation, and redirected policy updates; otherwise the anchors themselves can be subverted. Temporal compartmentalization likewise requires that TTL metadata be integrity- and freshness-protected at the memory side: host- or firmware-maintained TTL values could otherwise be extended, shortened,

rolled back, or left inconsistent across banks by an adversary operating within the capabilities defined in Section 6. The preliminary evaluation models representative maintenance traffic, but it does not implement these distributed protocols, memory-side metadata protection, or recovery paths.

- *Wear management and secure erasure*: In NVM-based PIM, secure erasure must coexist with wear management, address remapping, persistence, and recovery. Logical invalidation may leave residual copies in remapped physical locations. A practical solution requires controller-integrated sanitization, cryptographic erasure, or verified mapping-aware erasure; the framework identifies this requirement but does not define such a mechanism.
- *Side-channel protection and performance*: The simulated padding mechanism is configurable traffic shaping rather than a formal ORAM construction. It quantifies DRAM-interface performance sensitivity under the modeled workloads but does not establish an information-theoretic bound on access-pattern leakage. More generally, padding, isolation, monitoring, and scheduling restrictions may trade throughput and utilization for reduced leakage. Full-system multi-bank and multi-tenant evaluation is needed to characterize this trade-off and adaptive-policy stability.

These limitations expose a trust-transition gap. The framework declares the host untrusted yet does not specify how security-critical state (context assignments, TTL metadata, anchor credentials) is established and maintained without host involvement. Memory-side enforcement requires that these operations be verifiable independently of the host, but the interfaces and protocols for achieving this remain unspecified. Resolving this gap is the central prerequisite for the implementation and validation agenda outlined in the preceding roadmap.

8. Open Challenges and Concluding Outlook for Secure PIM Architectures

This section identifies the principal challenges that remain before PIM security mechanisms can move from proposal to deployment, drawing on the systemic gaps catalogued in Section 5.7 and the open design questions raised in Section 7. Two methodological limitations should be noted: screening was performed by a single primary reviewer (with co-author review of synthesis decisions and transparent reporting of all criteria, as described in Section 2), and no formal risk-of-bias instrument was applied, consistent with qualitative synthesis of heterogeneous architecture literature where standardized bias tools are not established. With these caveats in mind, the following gaps stand out as the most pressing barriers to secure PIM deployment.

- *Absence of a cross-layer, PIM-specific threat model*: The most consequential limitation identified across the surveyed literature is the absence of a standardized adversary model that spans host firmware, PCIe/CXL semantics, subarray-level fault phenomena, and non-volatile memory (NVM) persistence paths simultaneously. Existing CPU- or TEE-based adversary assumptions inadequately capture vectors such as operand manipulation, cross-bank leakage, and timing anomalies arising from the microarchitecture of near-memory compute. The four-class adversary model formalized in Section 6, covering the malicious co-tenant, compromised host OS or hypervisor, malicious firmware, and limited physical adversary, and its STRIDE mapping in Table 6 represent a step toward closing this gap. As Table 4 confirms, however, no surveyed mechanism addresses cross-layer security budget composition under multi-tenant conditions. A formal, shared threat-model specification, usable as a common reference baseline across hardware vendors, compiler designers, and certification bodies, is a prerequisite that the surveyed literature does not currently provide.

- *Multi-tenant isolation without physical-layer enforcement*: As PIM units grow more programmable and widely deployed in cloud and AI platforms, maintaining predictable boundaries between tenants, kernels, and memory banks becomes increasingly complex. Coarse-grained partitioning, as in NeuroPIM's vault-level static assignment [55], and software-only scheduling, as in Virtual PIM's DPU migration [56], reduce co-residency duration but cannot eliminate physical-layer coupling through shared sense amplifiers and refresh circuitry, a limitation each work acknowledges explicitly. As demonstrated by IMPACT, adversarial workloads can exploit exactly the mapping and scheduling freedoms these schemes rely upon to establish covert channels reaching 14.8 Mb/s [36]. The structural root cause, identified in Section 5.3, is the absence of hardware-enforced bank-level access quotas and per-tenant refresh isolation in current PIM architectures. No published work, to our knowledge, has proposed these mechanisms for PIM deployments; they remain open problems that must be resolved before secure multi-tenant operation is achievable at cloud scale.
- *Scalable, memory-native attestation*: Conventional TPM or enclave attestation mechanisms rely on centralized, host-visible control logic whose trust assumptions are fundamentally misaligned with distributed PIM arrays spanning banks, dies, and channels. SE-PIM and SecNDP demonstrate functional TEE-PIM integration [52,53], but their static enclave hierarchies depend on trusted fabrication and single-vendor root-of-trust provisioning that conflict with vendor-disaggregated CXL-attached deployments. Toleo addresses memory freshness across a 28 TB CXL-expanded pool [54] but does not address attestation of PIM-internal compute authenticity at bank granularity. The distributed trust-anchor model proposed in Section 7 addresses this structurally by instantiating per-bank verifiers with horizontal and vertical attestation chains, targeting attestation completion within the ≈35 ns HBM2e row-activation window established in Section 7 as the design bound. Lightweight, low-latency attestation protocols capable of operating within tight area and power budgets across disaggregated memory hierarchies remain a specific gap that neither the surveyed literature nor the proposed framework has yet closed empirically.
- *Formal verification tooling for PIM security*: Formal verification for PIM security remains immature. Modelling the interplay of side channels, transient states, DRAM timing behavior, and near-bank logic requires symbolic, formal, and fuzz-based frameworks that operate across architectural layers simultaneously. Current evaluation methodologies are inconsistent across the surveyed works: LT-PIM and P-PIM use overlapping but non-identical toolchains, rendering their overhead figures non-commensurable and indicative rather than directly comparable [12,13]. No existing framework verifies the two security properties stated as design requirements in Section 7, isolation invariance and TTL erasure completeness, across a full PIM defense stack. Without such tooling, certifying leakage bounds, isolation guarantees, and data-lifecycle integrity across disaggregated deployments remains infeasible, directly blocking certified deployment in regulated environments.
- *Security-aware compilers and programming models*: As established in Section 5.7, PIM security cannot be reasoned about at the hardware level alone. The toolchain determines which tenants share banks, subarrays, and row buffers, yet current PIM compilers and SDKs treat partitioning, obliviousness, and data-lifecycle constraints as incidental rather than as hard optimization constraints. Enclave-oriented designs such as SE-PIM and SecNDP depend on compilers preserving oblivious access patterns and avoiding unprotected temporaries [52,53]. Remanence-oriented designs such as SecPM and Triad-NVM assume that erase and persist operations are not reordered or eliminated during lowering into memory-resident kernels [64,66]. No current PIM compiler or

runtime exposes annotations for these properties or verifies that they survive the lowering process. Defining and enforcing these constraints at the compiler level is a prerequisite for composable, provable PIM security that the broader PIM toolchain community has not yet addressed.

8.1. Research Trajectories

The five challenges above point to four specific research problems, each grounded in specific limitations identified across this survey. The first two trajectories are short-term engineering goals achievable with existing simulation infrastructure and standards processes (1–3 year horizon); the latter two require longer-term architectural shifts involving cross-industry coordination and fundamental algorithmic advances (3–7 year horizon).

- *Threat-aware architectural design*: PIM-specific adversary models, formalized across the four attacker classes in Table 5 and mapped to STRIDE categories in Table 6, must be embedded into early architectural exploration rather than applied post-hoc against already-fixed microarchitectures, as is the pattern across the surveyed literature. This requires open RTL references, PIM-aware fuzz harnesses, and simulators such as DRAMSim3 coupled with RISC-V PIM ISA extensions that expose subarray-granular timing and activation behavior to security analysis. A measurable success criterion for this trajectory is the ability to verify, at the RTL level, the isolation invariance and TTL erasure completeness properties defined in Section 7, using symbolic model checkers such as CoSA or SymbiYosys before tape-out rather than through post-silicon adversarial evaluation. PIM security properties span multiple abstraction layers at once: a symbolic model checker must reason about DRAM timing constraints, digital logic correctness, and cross-bank interference effects in a single verification pass, which exceeds the capacity of existing single-layer formal tools. Whether subarray-level isolation invariants can even be expressed in a temporal logic that current hardware model checkers support is unclear. A related barrier is identifying the minimal set of RTL observability hooks that would expose PIM-internal activation patterns to a security fuzzer without degrading memory bandwidth.
- *Standardized attestation and secure execution interfaces*: As commercial PIM products mature, vendor-agnostic attestation interfaces at bank, row, and logic-cluster granularity are needed to enable independent verification without requiring access to proprietary controller microcode. The distributed trust-anchor model of Section 7 proposes one architectural basis for such interfaces, with attestation targeting completion within the ≈35 ns HBM2e row-activation window established there as the design bound. The interface specification, command-queue protocol, and failure semantics identified as open design work in Section 7 require formal definition before standardization is viable. Existing memory interface standardization efforts, including JEDEC's security metadata extensions and CXL's integrity and data encryption (IDE) protocol [45], provide coordination venues that PIM attestation standardization could build upon. Attestation must complete within tight memory protocol timing budgets (≈35 ns HBM2e row-activation window), yet the cryptographic operations it requires typically take orders of magnitude longer on conventional logic. The minimum viable cryptographic primitive for this setting, whether truncated HMAC, PUF-based challenge-response, or something not yet proposed, is unknown. Certificate revocation across independently manufactured memory modules in a CXL-attached topology presents a separate unsolved problem: propagation delays must not exceed the memory access latency budget, but no published mechanism achieves this.
- *AI-aware and multi-tenant security models*: Dynamic co-location of AI inference workloads with general-purpose tenants on shared PIM hardware creates isolation require-

ments that static partitioning cannot satisfy. Model-weight remanence, addressed partially by SFGE for NVM-based PIM [65], and inference-time access-pattern leakage, demonstrated by IMPACT's 7.6 Mb/s genomic side channel at 96% accuracy [36], represent threat classes that grow more severe as PIM adoption in AI acceleration expands. Security models must cover data and model privacy as well as execution integrity under dynamic workload co-location. The data-oblivious execution mechanism of Section 7 frames the measurable target for this trajectory as minimizing the mutual information between the observable memory access sequence and the sensitive operand value, subject to available memory bandwidth constraints, with policies that adapt at the granularity of individual inference passes rather than at static provisioning time. In a PIM simulator, this quantity could be estimated empirically by collecting bank-conflict timing distributions across a representative workload sweep, then applying a k-nearest-neighbor mutual information estimator to the resulting access-pattern traces, a methodology established for cache side-channel analysis and directly portable to the row-buffer timing model. The design target is to reduce this estimated mutual information below a threshold that renders timing-based inference computationally infeasible for a co-resident adversary with realistic sampling bandwidth. AI inference workloads exhibit highly structured, model-dependent access patterns that are far more predictable than general-purpose computation, which makes obfuscation both more necessary and more expensive. Whether workload-adaptive obfuscation can reduce mutual information enough to matter without consuming so much bandwidth that PIM loses its advantage over CPU baselines is the central unresolved tension. A related problem is the update granularity of isolation policies: tracking dynamic inference-phase transitions in multi-tenant deployments requires policy updates at a frequency that has not been characterized, and too-frequent updates risk introducing overhead that rivals the leakage they prevent.

- *Ecosystem coordination across hardware, software, and certification bodies*: The fragmentation identified in Section 5.7, defenses siloed across host–PIM orchestration, isolation, side-channel resilience, remanence, and reliability without a shared composition framework, cannot be resolved by individual research contributions alone. Hardware vendors, compiler and OS communities, memory standards bodies, and certification authorities must coordinate on a shared security budget framework that quantifies how area, timing, and bandwidth overheads aggregate when multiple defenses are deployed simultaneously. The quantitative overhead data synthesized in Table 2 provides a concrete starting point: in-DRAM RowHammer tracking imposes 0.14–0.8% runtime overhead [12,13]; orchestration verification adds under 2% [47]; and memory-bus side-channel protection incurs approximately 20.74% performance overhead [59]. End-to-end composition of these overheads across a full defense stack has not been evaluated by any published work and constitutes the most immediate gap between current research and certified deployment. Defense overheads may not compose linearly: encryption-induced bandwidth pressure can amplify timing-channel exposure, isolation policies may intensify disturbance escalation, and monitoring frequency trades off against the very contention it is designed to detect. The preliminary composition experiments in Table 11 are encouraging, showing near-linear interaction (factors of 1.00–1.02) for three mechanisms at the DRAM interface level, but this covers only proxy-level overhead modeling on a single-bank configuration. It is unknown whether near-linear composition holds under more realistic conditions with hardware-enforced isolation logic, multi-bank parallelism, and concurrent host-PIM traffic. Independently, the ecosystem needs a shared security-budget specification that lets vendors certify

their defense contributions without full-stack integration testing, but defining such a specification requires agreement on composition semantics that do not yet exist.

8.2. Conclusions

This survey addresses the gap identified in Section 1: no prior work consolidates PIM security threats, attacker capabilities, and architectural countermeasures under a unified analytical structure applicable across all three PIM architecture classes, Logic-in-Memory (LiM), Near-Data Processing (NDP), and NVM-based PIM. Three findings emerge from this synthesis. First, the structured adversary model of Section 6 formalizes four attacker classes with explicit capability boundaries and STRIDE-mapped success conditions across the host–PIM interface, PIM logic microarchitecture, DRAM substrate, and NVM persistence layer. The threat surface is cross-layer by construction: no single-boundary defense is sufficient, and the interaction effects between defenses at adjacent layers remain uncharacterized in the literature. Second, the six-domain threat classification and quantitative defense synthesis of Section 5 demonstrate that existing mechanisms address narrow threat axes without converging into a composable defense stack. No published work has measured the aggregate overhead of deploying multiple defenses simultaneously: performance overheads ranging from under 2% for orchestration verification [47] to approximately 20.74% for bus-level address protection [59] have been measured in isolation. The preliminary composition experiments in Table 11 provide the first such measurement, showing near-linear interaction (factors of 1.00–1.02) across three mechanisms at the DRAM interface level, though full-system composition under realistic multi-bank conditions remains uncharacterized. Third, the conceptual secure-by-design framework of Section 7 proposes five composable, memory-resident mechanisms, subarray-level isolation, temporal compartmentalization, distributed trust anchoring, data-oblivious execution, and adaptive policy orchestration, each targeting a distinct axis of the threat environment and each paired with the specific validation track required before implementation claims can be made. The central structural argument is that PIM security requires trust enforcement located within the memory substrate itself, with host-side software obtaining attestation from memory rather than imposing it upon memory. A boundary condition applies: as detailed in Table 9, the five mechanisms are directly applicable to NDP-class architectures with programmable digital controllers, applicable at controller granularity for DRAM-based LiM designs, and insufficient by themselves for purely analog compute-in-memory substrates where leakage originates within the analog array. For hybrid architectures combining analog crossbars with digital controllers, the framework covers the digital side of the ADC/DAC interface; crossbar-internal leakage requires device- and mixed-signal-level protection, as discussed in Section 7. As PIM architectures move from research prototypes toward production deployment in cloud and AI platforms, the security gap between what these systems can do and what they can do safely will determine whether PIM fulfills its architectural promise or introduces systemic risk at scale. The adversary model, threat taxonomy, and composable framework presented here provide an analytical foundation for closing that gap.

Supplementary Materials: The following supporting information can be downloaded at https://www.mdpi.com/article/10.3390/computers15090611/s1, Supplementary File S1: PRISMA 2020 checklist for the systematic review.

Author Contributions: Conceptualization, S.W.; methodology, S.W.; investigation, S.W.; formal analysis, S.W.; writing—original draft preparation, S.W.; writing—review and editing, S.W., H.Q.L. and D.B.; visualization, S.W.; supervision, H.Q.L. and D.B. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: No new data were created or analyzed in this study. Data sharing is not applicable to this article.

Acknowledgments: During the preparation of this manuscript/study, the authors used Claude (https://claude.ai) for the purposes of LaTeX formatting assistance, including table layout conversion and language polishing and grammar refinement of the manuscript text. The authors have reviewed and edited the output and take full responsibility for the content of this publication.

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

PIM	Processing-in-Memory
CPU	Central Processing Unit
GPU	Graphics Processing Unit
AI	Artificial Intelligence
LLM	Large Language Model
DRAM	Dynamic Random-Access Memory
NVM	Non-Volatile Memory
TRR	Target Row Refresh
DDR4	Double Data Rate 4
NDP	Near-Data Processing
LiM	Logic-in-Memory
PCIe	Peripheral Component Interconnect Express
CXL	Compute Express Link
DMA	Direct Memory Access
ReRAM	Resistive Random-Access Memory
PCM	Phase-Change Memory
STT-RAM	Spin-Transfer Torque Random-Access Memory
MVM	Matrix–Vector Multiplication
HBM	High Bandwidth Memory
SIMD	Single Instruction, Multiple Data
MMU	Memory Management Unit
TEE	Trusted Execution Environment
SGX	Software Guard Extensions
MPC	Multi-Party Computation
ORAM	Oblivious Random-Access Memory
DPU	Data Processing Unit
ECC	Error-Correcting Code
MRAM	Magnetoresistive Random-Access Memory
DLRM	Deep Learning Recommendation Model
MLP	Multi-Layer Perceptron
TTL	Time-to-Live
RTL	Register Transfer Level
JEDEC	Joint Electron Device Engineering Council
IDE	Integrity and Data Encryption
PRISMA	Preferred Reporting Items for Systematic Reviews and Meta-Analyses
STRIDE	Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, Elevation of Privilege
TLB	Translation Lookaside Buffer

IPC	Instructions Per Cycle
FPGA	Field-Programmable Gate Array
SDK	Software Development Kit
API	Application Programming Interface
PKI	Public Key Infrastructure
TPM	Trusted Platform Module
OS	Operating System

References

1. Hennessy, J.L.; Patterson, D.A. A new golden age for computer architecture. *Commun. ACM* **2019**, *62*, 48–60. [CrossRef] [PubMed]
2. Wulf, W.A.; McKee, S.A. Hitting the memory wall: Implications of the obvious. *ACM SIGARCH Comput. Archit. News* **1995**, *23*, 20–24. [CrossRef]
3. Boroumand, A.; Ghose, S.; Kim, Y.; Ausavarungnirun, R.; Shiu, E.; Thakur, R.; Kim, D.; Kuusela, A.; Knies, A.; Ranganathan, P.; et al. Google Workloads for Consumer Devices: Mitigating Data Movement Bottlenecks. In Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems, Williamsburg, VA, USA, 24–28 March 2018; pp. 316–331. [CrossRef]
4. Mutlu, O.; Olgun, A.; Oliveira, G.F.; Yuksel, I.E. Memory-Centric Computing: Recent Advances in Processing-in-DRAM (Invited). In *Proceedings of the 2024 IEEE International Electron Devices Meeting (IEDM)*; IEEE: San Francisco, CA, USA, 2024; pp. 1–4. [CrossRef]
5. Mutlu, O.; Ghose, S.; Gómez-Luna, J.; Ausavarungnirun, R. Enabling Practical Processing in and near Memory for Data-Intensive Computing. In *Proceedings of the 56th Annual Design Automation Conference 2019 (DAC '19)*; ACM: New York, NY, USA, 2019; pp. 1–4. [CrossRef]
6. Kwon, Y.; Lee, Y.; Rhu, M. 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, Columbus, OH, USA, 12–16 October 2019; pp. 740–753. [CrossRef]
7. Cai, S.; Tian, B.; Zhang, H.; Gao, M. PimPam: Efficient Graph Pattern Matching on Real Processing-in-Memory Hardware. *Proc. ACM Manag. Data* **2024**, *2*, 1–25. [CrossRef]
8. Ortega, C.; Falevoz, Y.; Ayrignac, R. PIM-AI: A Novel Architecture for High-Efficiency LLM Inference. *arXiv* **2024**, arXiv:2411.17309.
9. Lee, S.; Kang, S.h.; Lee, J.; Kim, H.; Lee, E.; Seo, S.; Yoon, H.; Lee, S.; Lim, K.; Shin, H.; et al. Hardware Architecture and Software Stack for PIM Based on Commercial DRAM Technology: Industrial Product. In Proceedings of the 2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA), Valencia, Spain, 14–18 June 2021; pp. 43–56. [CrossRef]
10. Devaux, F. The True Processing-In-Memory (PIM) Accelerator. In *Proceedings of the 2019 IEEE Hot Chips 31 Symposium (HCS)*; IEEE: Cupertino, CA, USA, 2019; pp. 1–24. [CrossRef]
11. Jattke, P.; Van Der Veen, V.; Frigo, P.; Gunter, S.; Razavi, K. BLACKSMITH: Scalable Rowhammering in the Frequency Domain. In Proceedings of the 2022 IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA, 22–26 May 2022; pp. 716–734. [CrossRef]
12. Zhou, R.; Tabrizchi, S.; Roohi, A.; Angizi, S. LT-PIM: An LUT-Based Processing-in-DRAM Architecture With RowHammer Self-Tracking. *IEEE Comput. Archit. Lett.* **2022**, *21*, 141–144. [CrossRef]
13. Zhou, R.; Tabrizchi, S.; Morsali, M.; Roohi, A.; Angizi, S. P-PIM: A Parallel Processing-in-DRAM Framework Enabling Row Hammer Protection. In Proceedings of the 2023 Design, Automation & Test in Europe Conference & Exhibition (DATE), Antwerp, Belgium, 17–19 April 2023; pp. 1–6. [CrossRef]
14. Chi, P.; Li, S.; Xu, C.; Zhang, T.; Zhao, J.; Liu, Y.; Wang, Y.; Xie, Y. PRIME: A Novel Processing-in-Memory Architecture for Neural Network Computation in ReRAM-Based Main Memory. In Proceedings of the 2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA), Seoul, Republic of Korea, 18–22 June 2016; pp. 27–39. [CrossRef]
15. Pajouhi, Z.; Fong, X.; Raghunathan, A.; Roy, K. Yield, Area, and Energy Optimization in STT-MRAMs Using Failure-Aware ECC. *ACM J. Emerg. Technol. Comput. Syst. (JETC)* **2016**, *13*, 1–20. [CrossRef]
16. Mutlu, O.; Kim, J.S. RowHammer: A Retrospective. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **2020**, *39*, 1555–1571. [CrossRef]
17. Arafin, M.T.; Lu, Z. Security Challenges of Processing-In-Memory Systems. In Proceedings of the 2020 Great Lakes Symposium on VLSI, Virtual Event China, 7–9 September 2020; pp. 229–234. [CrossRef]
18. Page, M.J.; McKenzie, J.E.; Bossuyt, P.M.; Boutron, I.; Hoffmann, T.C.; Mulrow, C.D.; Shamseer, L.; Tetzlaff, J.M.; Akl, E.A.; Brennan, S.E.; et al. The PRISMA 2020 statement: An updated guideline for reporting systematic reviews. *BMJ* **2021**, *372*, n71. [CrossRef] [PubMed]

19. Ben-Hur, R.; Ronen, R.; Haj-Ali, A.; Bhattacharjee, D.; Eliahu, A.; Peled, N.; Kvatinsky, S. SIMPLER MAGIC: Synthesis and Mapping of In-Memory Logic Executed in a Single Row to Improve Throughput. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **2020**, *39*, 2434–2447. [CrossRef]
20. Li, S.; Niu, D.; Malladi, K.T.; Zheng, H.; Brennan, B.; Xie, Y. DRISA: A DRAM-based Reconfigurable In-Situ Accelerator. In Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture, Cambridge, MA, USA, 14–18 October 2017; pp. 288–301. [CrossRef]
21. Shafiee, A.; Nag, A.; Muralimanohar, N.; Balasubramonian, R.; Strachan, J.P.; Hu, M.; Williams, R.S.; Srikumar, V. ISAAC: A Convolutional Neural Network Accelerator with In-Situ Analog Arithmetic in Crossbars. In Proceedings of the 2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA), Seoul, Republic of Korea, 18–22 June 2016; pp. 14–26. [CrossRef]
22. Seshadri, V.; Lee, D.; Mullins, T.; Hassan, H.; Boroumand, A.; Kim, J.; Kozuch, M.A.; Mutlu, O.; Gibbons, P.B.; Mowry, T.C. Ambit: In-memory accelerator for bulk bitwise operations using commodity DRAM technology. In Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture, Cambridge, MA, USA, 14–18 October 2017; pp. 273–287. [CrossRef]
23. Seshadri, V.; Mutlu, O. In-DRAM Bulk Bitwise Execution Engine. *arXiv* **2020**, arXiv:1905.09822.
24. Kim, Y.; Daly, R.; Kim, J.; Fallin, C.; Lee, J.H.; Lee, D.; Wilkerson, C.; Lai, K.; Mutlu, O. Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors. In Proceedings of the 2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA), Minneapolis, MN, USA, 14–18 June 2014; pp. 361–372. [CrossRef]
25. Jiang, Y.; Zhu, H.; Sullivan, D.; Guo, X.; Zhang, X.; Jin, Y. Quantifying Rowhammer Vulnerability for DRAM Security. In Proceedings of the 2021 58th ACM/IEEE Design Automation Conference (DAC), San Francisco, CA, USA, 5–9 December 2021; pp. 73–78. [CrossRef]
26. Ahn, J.; Hong, S.; Yoo, S.; Mutlu, O.; Choi, K. A scalable processing-in-memory accelerator for parallel graph processing. In Proceedings of the 42nd Annual International Symposium on Computer Architecture, Portland, OR, USA, 13–17 June 2015; pp. 105–117. [CrossRef]
27. Naghibijouybari, H.; Koruyeh, E.M.; Abu-Ghazaleh, N. Microarchitectural Attacks in Heterogeneous Systems: A Survey. *ACM Comput. Surv.* **2023**, *55*, 1–40. [CrossRef]
28. Burr, G.W.; Shelby, R.M.; Sebastian, A.; Kim, S.; Kim, S.; Sidler, S.; Virwani, K.; Ishii, M.; Narayanan, P.; Fumarola, A.; et al. Neuromorphic computing using non-volatile memory. *Adv. Phys. X* **2017**, *2*, 89–124. [CrossRef]
29. Chen, I.Y.; Hou, K.W.; Chen, Y.S. A Configurable ReRAM Engine for Energy-Efficient Sparse Neural Network Acceleration. *IEEE Embed. Syst. Lett.* **2025**, *17*, 305–308. [CrossRef]
30. Geng, B.; Fan, M.; Jin, Z.; Liu, W. ReRAM-Based Process-In-Memory Accelerator for Iterative Solvers: A Systematic Survey. In Proceedings of the 2025 IEEE International Symposium on Circuits and Systems (ISCAS), London, UK, 25–28 May 2025; pp. 1–5. [CrossRef]
31. Li, B.; Yan, B.; Li, H. An Overview of In-memory Processing with Emerging Non-volatile Memory for Data-intensive Applications. In *Proceedings of the 2019 Great Lakes Symposium on VLSI (GLSVLSI '19)*; ACM: Tysons Corner, VA, USA, 2019; pp. 381–386. [CrossRef]
32. Liu, C.; Wu, K.; Liu, H.; Jin, H.; Liao, X.; Duan, Z.; Xu, J.; Li, H.; Zhang, Y.; Yang, J. A ReRAM-Based Processing-In-Memory Architecture for Hyperdimensional Computing. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **2025**, *44*, 512–524. [CrossRef]
33. Mohammadi, A.; Cheshmikhani, E.; Asadi, H. A Reliability-Aware Replacement Policy for STT-MRAM Caches in Server-Class Processors. *IEEE Trans. Reliab.* **2025**, *74*, 4915–4929. [CrossRef]
34. Peng, X.; Huang, S.; Jiang, H.; Lu, A.; Yu, S. DNN+NeuroSim V2.0: An End-to-End Benchmarking Framework for Compute-in-Memory Accelerators for On-Chip Training. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **2021**, *40*, 2306–2319. [CrossRef]
35. Pan, Y.; Ouyang, P.; Zhao, Y.; Kang, W.; Yin, S.; Zhang, Y.; Zhao, W.; Wei, S. A Multilevel Cell STT-MRAM-Based Computing In-Memory Accelerator for Binary Convolutional Neural Network. *IEEE Trans. Magn.* **2018**, *54*, 1–5. [CrossRef]
36. Bostancı, F.N.; Kanellopoulos, K.; Olgun, A.; Yağlıkçı, A.G.; Yüksel, I.E.; Mansouri Ghiasi, N.; Bingöl, Z.; Sadrosadati, M.; Mutlu, O. Revisiting Main Memory-Based Covert and Side Channel Attacks in the Context of Processing-in-Memory. In *Proceedings of the 2025 55th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*; IEEE: Naples, Italy, 2025; pp. 16–32. [CrossRef]
37. Mutlu, O. The RowHammer problem and other issues we may face as memory becomes denser. In Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017, Lausanne, Switzerland, 27–31 March 2017; pp. 1116–1121. [CrossRef]
38. Mishra, J.; Sahay, S.K. Modern Hardware Security: A Review of Attacks and Countermeasures. *arXiv* **2025**, arXiv:2501.04394.
39. Duy, K.D.; Lee, H. PIM-Enclave: Bringing Confidential Computation Inside Memory. *arXiv* **2021**, arXiv:2111.03307.

40. Hyun, B.; Kim, T.; Lee, D.; Rhu, M. Pathfinding Future PIM Architectures by Demystifying a Commercial PIM Technology. In Proceedings of the 2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA), Edinburgh, UK, 2–6 March 2024; pp. 263–279. [CrossRef]
41. Loughlin, K.; Rosenblum, J.; Saroiu, S.; Wolman, A.; Skarlatos, D.; Kasikci, B. Siloz: Leveraging DRAM Isolation Domains to Prevent Inter-VM Rowhammer. In Proceedings of the 29th Symposium on Operating Systems Principles, Koblenz, Germany, 23–26 October 2023; pp. 417–433. [CrossRef]
42. Checkoway, S.; Shacham, H. Iago attacks: Why the system call API is a bad untrusted RPC interface. In *Proceedings of the 18th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '13)*; ACM: Houston, TX, USA, 2013; pp. 253–264. [CrossRef]
43. Mwaisela, M. PhD Forum: Efficient Privacy-Preserving Processing via Memory-Centric Computing. In *Proceedings of the 2024 43rd International Symposium on Reliable Distributed Systems (SRDS)*; IEEE: Charlotte, NC, USA, 2024; pp. 322–325. [CrossRef]
44. Ahn, J.; Yoo, S.; Mutlu, O.; Choi, K. PIM-enabled instructions: A low-overhead, locality-aware processing-in-memory architecture. In Proceedings of the 42nd Annual International Symposium on Computer Architecture, Portland, OR, USA, 13–17 June 2015; pp. 336–348. [CrossRef]
45. Das Sharma, D.; Blankenship, R.; Berger, D. An Introduction to the Compute Express Link (CXL) Interconnect. *ACM Comput. Surv.* **2024**, *56*, 1–37. [CrossRef]
46. Boroumand, A.; Ghose, S.; Patel, M.; Hassan, H.; Lucia, B.; Hsieh, K.; Malladi, K.T.; Zheng, H.; Mutlu, O. LazyPIM: An Efficient Cache Coherence Mechanism for Processing-in-Memory. *IEEE Comput. Archit. Lett.* **2017**, *16*, 46–50. [CrossRef]
47. Bolat, A.; Tuğrul, Y.C.; Çelik, S.H.; Sezer, S.; Ottavi, M.; Ergin, O. DEV-PIM: Dynamic Execution Validation with Processing-in-Memory. In Proceedings of the 2023 IEEE European Test Symposium (ETS), Venezia, Italy, 22–26 May 2023; pp. 1–6. [CrossRef]
48. Lebedev, I.; Hogan, K.; Drean, J.; Kohlbrenner, D.; Lee, D.; Asanovic, K.; Song, D.; Devadas, S. Sanctorum: A lightweight security monitor for secure enclaves. In Proceedings of the 2019 Design, Automation & Test in Europe Conference & Exhibition (DATE), Florence, Italy, 25–29 March 2019; pp. 1142–1147. [CrossRef]
49. Lee, D.; Kohlbrenner, D.; Shinde, S.; Asanović, K.; Song, D. Keystone: An open framework for architecting trusted execution environments. In *Proceedings of the 15th European Conference on Computer Systems (EuroSys '20)*; ACM: Heraklion, Greece, 2020; pp. 1–16. [CrossRef]
50. Ngabonziza, B.; Martin, D.; Bailey, A.; Cho, H.; Martin, S. TrustZone Explained: Architectural Features and Use Cases. In Proceedings of the 2016 IEEE 2nd International Conference on Collaboration and Internet Computing (CIC), Pittsburgh, PA, USA, 1–3 November 2016; pp. 445–451. [CrossRef]
51. Nilsson, A.; Bideh, P.N.; Brorsson, J. A Survey of Published Attacks on Intel SGX. *arXiv* **2020**, arXiv:2006.13598.
52. Duy, K.D.; Lee, H. SE-PIM: In-Memory Acceleration of Data-Intensive Confidential Computing. *IEEE Trans. Cloud Comput.* **2023**, *11*, 2473–2490. [CrossRef]
53. Xiong, W.; Ke, L.; Jankov, D.; Kounavis, M.; Wang, X.; Northup, E.; Yang, J.A.; Acun, B.; Wu, C.J.; Peter Tang, P.T.; et al. SecNDP: Secure Near-Data Processing with Untrusted Memory. In Proceedings of the 2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA), Seoul, Republic of Korea, 2–6 April 2022; pp. 244–258. [CrossRef]
54. Dong, J.; Rosenblum, J.; Narayanasamy, S. Toleo: Scaling Freshness to Tera-scale Memory Using CXL and PIM. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 4 (ASPLOS '25)*; ACM: La Jolla, CA, USA, 2025; pp. 1031–1046. [CrossRef]
55. Bidgoli, A.M.; Fattahi, S.; Rezaei, S.H.S.; Modarressi, M.; Daneshtalab, M. NeuroPIM: Flexible Neural Accelerator for Processing-in-Memory Architectures. In Proceedings of the 2023 26th International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS), Tallinn, Estonia, 3–5 May 2023; pp. 51–56. [CrossRef]
56. Kim, D.; Kim, T.; Hwang, I.; Park, T.; Kim, H.; Kim, Y.; Park, Y. Virtual PIM: Resource-Aware Dynamic DPU Allocation and Workload Scheduling Framework for Multi-DPU PIM Architecture. In Proceedings of the 2023 32nd International Conference on Parallel Architectures and Compilation Techniques (PACT), Vienna, Austria, 21–25 October 2023; pp. 112–123. [CrossRef]
57. Pessl, P.; Gruss, D.; Maurice, C.; Schwarz, M.; Mangard, S. DRAMA: Exploiting DRAM Addressing for Cross-CPU Attacks. In Proceedings of the 25th USENIX Security Symposium (USENIX Security 16), Austin, TX, USA, 10–12 August 2016; pp. 633–650. Available online: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/pessl (accessed on 7 September 2026).
58. Gómez-Luna, J.; El Hajj, I.; Fernandez, I.; Giannoula, C.; Oliveira, G.F.; Mutlu, O. Benchmarking a New Paradigm: Experimental Analysis and Characterization of a Real Processing-in-Memory System. *IEEE Access* **2022**, *10*, 52565–52608. [CrossRef]
59. Aga, S.; Narayanasamy, S. InvisiMem: Smart Memory Defenses for Memory Bus Side Channel. In Proceedings of the 44th Annual International Symposium on Computer Architecture, Toronto, ON, Canada, 24–28 June 2017; pp. 94–106. [CrossRef]

60. Woo, B.; Duy, K.D.; Han, Y.; Kang, B.B.; Lee, H. PIM-ORAM: Towards Oblivious RAM Primitives in Commodity Processing-In-Memory. In Proceedings of the 2025 IEEE Annual Computer Security Applications Conference (ACSAC), Honolulu, HI, USA, 8–12 December 2025; pp. 1018–1033. [CrossRef]
61. Wang, Z.; Taram, M.; Moghimi, D.; Swanson, S.; Tullsen, D.M.; Zhao, J. NVLeak: Off-Chip Side-Channel Attacks via Non-Volatile Memory Systems. In Proceedings of the 32nd USENIX Security Symposium (USENIX Security 23), Anaheim, CA, USA, 9–11 August 2023; pp. 1497–1514. Available online: https://www.usenix.org/conference/usenixsecurity23/presentation/wang-zixuan (accessed on 7 September 2026).
62. Xie, Z.; Han, Y.; Chen, X. PME: Processing-in-memory Masking and Encoding for Secure NVM. In Proceedings of the 2022 IEEE 24th Int Conf on High Performance Computing & Communications; 8th Int Conf on Data Science & Systems; 20th Int Conf on Smart City; 8th Int Conf on Dependability in Sensor, Cloud & Big Data Systems & Application (HPCC/DSS/SmartCity/DependSys), Hainan, China, 18–20 December 2022; pp. 1501–1508. [CrossRef]
63. Awad, A.; Manadhata, P.; Haber, S.; Solihin, Y.; Horne, W. Silent Shredder: Zero-Cost Shredding for Secure Non-Volatile Main Memory Controllers. In Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems, Atlanta, GA, USA, 2–6 April 2016; pp. 263–276. [CrossRef]
64. Zuo, P.; Hua, Y. SecPM: A Secure and Persistent Memory System for Non-volatile Memory. In *Proceedings of the 10th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 18)*; USENIX Association: Boston, MA, USA, 2018. Available online: https://www.usenix.org/conference/hotstorage18/presentation/zuo (accessed on 7 September 2026).
65. Cai, Y.; Chen, X.; Tian, L.; Wang, Y.; Yang, H. Enabling Secure in-Memory Neural Network Computing by Sparse Fast Gradient Encryption. In Proceedings of the 2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), Westminster, CO, USA, 4–7 November 2019; pp. 1–8. [CrossRef]
66. Awad, A.; Ye, M.; Solihin, Y.; Njilla, L.; Zubair, K.A. Triad-NVM: Persistency for Integrity-Protected and Encrypted Non-Volatile Memories. In *Proceedings of the 46th International Symposium on Computer Architecture (ISCA '19)*; ACM: Pheonix, AZ, USA, 2019; pp. 104–115. [CrossRef]
67. Freij, A.; Zhou, H.; Solihin, Y. SecPB: Architectures for Secure Non-Volatile Memory with Battery-Backed Persist Buffers. In Proceedings of the 2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA), Montreal, QC, Canada, 25 February–1 March 2023; pp. 677–690. [CrossRef]
68. Fakhrzadehgan, A.; Patt, Y.N.; Nair, P.J.; Qureshi, M.K. SafeGuard: Reducing the Security Risk from Row-Hammer via Low-Cost Integrity Protection. In Proceedings of the 2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA), Seoul, Republic of Korea, 2–6 April 2022; pp. 373–386. [CrossRef]
69. Cojocar, L.; Razavi, K.; Giuffrida, C.; Bos, H. Exploiting Correcting Codes: On the Effectiveness of ECC Memory Against Rowhammer Attacks. In Proceedings of the 2019 IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA, 19–23 May 2019; pp. 55–71. [CrossRef]
70. Frigo, P.; Vannacc, E.; Hassan, H.; Van Der Veen, V.; Mutlu, O.; Giuffrida, C.; Bos, H.; Razavi, K. TRRespass: Exploiting the Many Sides of Target Row Refresh. In Proceedings of the 2020 IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA, 18–21 May 2020; pp. 747–762. [CrossRef]
71. Ghinani, S.G.; Zhang, J.; Sadredini, E. Enabling Low-Cost Secure Computing on Untrusted In-Memory Architectures. In Proceedings of the 34th USENIX Security Symposium (USENIX Security 25), Seattle, WA, USA, 13–15 August 2025; pp. 1749–1767. Available online: https://www.usenix.org/conference/usenixsecurity25/presentation/ghinani (accessed on 7 September 2026).
72. Costan, V.; Lebedev, I.; Devadas, S. Sanctum: Minimal hardware extensions for strong software isolation. In *Proceedings of the 25th USENIX Conference on Security Symposium (SEC '16)*; USENIX Association: Austin, TX, USA, 2016; pp. 857–874. Available online: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/costan (accessed on 7 September 2026).
73. Stefanov, E.; van Dijk, M.; Shi, E.; Fletcher, C.; Ren, L.; Yu, X.; Devadas, S. Path ORAM: An extremely simple oblivious RAM protocol. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security (CCS '13)*; ACM: Berlin, Germany, 2013; pp. 299–310. [CrossRef]
74. Kraskov, A.; Stögbauer, H.; Grassberger, P. Estimating mutual information. *Phys. Rev. E* **2004**, *69*, 066138. [CrossRef] [PubMed]
75. Lowe-Power, J.; Ahmad, A.; Akram, A.; Alian, M.; Amslinger, R.; Andreozzi, M.; Beaumont, A.; Beshara, M.; Binkert, N.; Black, B.; et al. The gem5 Simulator: Version 20.0+. *arXiv* **2020**, arXiv:2007.03152. [CrossRef]