No One-Size-Fits-All: A Workload-Driven Characterization of Bit-Parallel vs. Bit-Serial Data Layouts for Processing-using-Memory

Jingyao Zhang, Elaheh Sadredini
Department of Computer Science and Engineering
University of California, Riverside

Abstract

Processing-in-Memory (PIM) is a promising approach to overcoming the memory-wall bottleneck. However, the PIM community has largely treated its two fundamental data layouts, Bit-Parallel (BP) and Bit-Serial (BS), as if they were interchangeable. This implicit "one-layout-fits-all" assumption, often hard-coded into existing evaluation frameworks, creates a critical gap: architects lack systematic, workload-driven guidelines for choosing the optimal data layout for their target applications. To address this gap, this paper presents the first systematic, workload-driven characterization of BP and BS PIM architectures. We develop iso-area, cycle-accurate BP and BS PIM architectural models and conduct a comprehensive evaluation using a diverse set of benchmarks. Our suite includes both fine-grained microworkloads from MIMDRAM to isolate specific operational characteristics, and large-scale applications from the PIMBench suite, such as the VGG network, to represent realistic end-to-end workloads. Our results quantitatively demonstrate that no single layout is universally superior; the optimal choice is strongly dependent on workload characteristics. BP excels on control-flow-intensive tasks with irregular memory access patterns, whereas BS shows substantial advantages in massively parallel, low-precision (e.g., INT4/INT8) computations common in AI. Based on this characterization, we distill a set of actionable design guidelines for architects. This work challenges the prevailing one-size-fits-all view on PIM data layouts and provides a principled foundation for designing next-generation, workload-aware, and potentially hybrid PIM systems.

1. Introduction

The growing mismatch between processor speed and memory bandwidth, commonly referred to as the *memory wall*, has become one of the most pressing challenges in modern computing. As applications scale in data intensity, the cost of moving data between memory and computation units increasingly dominates overall system performance and energy consumption. Recent studies show that data movement can account for more than 60% of system energy in large-scale applications [5]. Processing-in-Memory (PIM) has emerged as a promising architectural approach to mitigate this challenge by relocating computation closer to where data resides.

PIM architectures can be broadly divided into two categories: Processing-Near-Memory (PNM) and Processing-Using-Memory (PUM). PNM places general-purpose or specialized compute logic near the memory periphery, either within memory modules, in the controller, or on the logic die in 3D-stacked DRAM, as demonstrated in academia (Chameleon [4], RecNMP [25], and Tesseract [27]) and commercial systems (Samsung's HBM-PIM [22] and UPMEM [11]). These systems benefit from reduced memory access latency and high bandwidth but still suffer from limited internal memory parallelism. In contrast, PUM architectures exploit the physical properties of memory cells to compute *in situ*, directly within the memory arrays. By turning memory into a compute substrate, PUM systems can eliminate processor-memory communication overhead and enable massive parallelism and energy efficiency. DRAM-based PUM systems such as Ambit [31], DRISA [24], pLUTo [13], and PULSAR [39] demonstrate how bulk bitwise operations and table lookups can be efficiently executed in commodity DRAM. Similarly, SRAM-based designs like Neural Cache [12] and Compute Caches [1] enable reconfigurable or bit-serial logic execution directly in SRAM arrays. Non-volatile memory (NVM) technologies, such as ReRAM and PCM, also support compute-in-memory through techniques like PRIME [8] and Pinatubo [23], which exploit resistive switching to perform parallel logic operations.

While PUM architectures have made significant progress, one critical but underexplored design choice in PUM remains the organization of data within memory arrays. At the heart of every PUM system is a fundamental decision: whether to adopt a **Bit-Parallel (BP)** layout, in which multi-bit words are stored horizontally across multiple bitlines to support word-level parallelism; or a **Bit-Serial (BS)** layout, where bits of a word are stored vertically along a single column, enabling massive bit-level parallelism. This choice has profound implications for performance and resource utilization.

Many works adopt layout strategies driven by specific hardware constraints or workload assumptions, without systematic justification. For example, BitFusion [32] adopts a BS layout to exploit fine-grained bit-level reuse for DNN inference, while PRIME [9] uses a BP layout to enable analog matrix-vector multiplication in ReRAM crossbars. Ambit [31] supports both layouts depending on the type of operation but offers no framework for guiding layout selection. Other architectures like BP-NTT [40], DRISA [24], Cascade [10], and FloatPIM [17] also make fixed layout choices

without articulating how these decisions affect broader system behavior across workloads. In many cases, these decisions are optimized for a narrow class of operations or tied to hardware capabilities, with little discussion of trade-offs or alternatives. Even comprehensive surveys [26, 35] that aim to organize the PIM design space treat data layout as a backend implementation detail, overlooking its first-order effects on programmability, control overhead, data reuse, and hardware efficiency.

Benchmarking tools further reinforce this oversight: for instance, PIMeval [33], a state-of-the-art benchmarking framework for PIM, statically associates data layout with hardware templates; for example, selecting a BitSIMD-V device enforces the use of a bit-serial (vertical) layout for all operations, irrespective of the specific workload characteristics. This rigid assumption fails to reflect the diverse computational needs of real-world applications and conceals potential inefficiencies introduced by mismatched layout-workload pairings. To illustrate the impact of this assumption, we conduct an evaluation of VGG-13 inference on a 512-column SRAM-based PIM array configured with a bit-serial layout. In the fully connected layers, where only 8 output neurons are active, the limited degree of parallelism results in severe underutilization—only 5.5% of the available compute columns are used. This inefficiency leads to wasted cycles and elevated energy per operation, underscoring the need for layout-aware workload mapping.

In summary, while existing PUM systems implement a range of data layout strategies, they largely lack systematic analysis or data-driven guidance for layout selection. This has fostered a “one-size-fits-all” mindset, where layout choices are fixed without considering workload-specific requirements, leading to inefficiencies and suboptimal designs. This implicit assumption reveals itself in three key limitations. First, despite numerous proposals, no systematic study has characterized how BP and BS layouts differ in their fundamental capabilities. For instance, while bit-serial excels at massively parallel bit-wise operations common in binary neural networks [30], it suffers from severe row overflow when storing multiple word-level intermediate results, a common pattern in iterative algorithms like FIR (Finite Impulse Response) filters. Conversely, bit-parallel naturally handles such scenarios with superior efficiency. However, when workloads exhibit massive bit-level parallelism that exceeds the array’s PE count, such as computing hamming distances across thousands of binary vectors, bit-serial’s ability to use every column as an independent 1-bit ALU provides a fundamental advantage. Second, existing PIM evaluations typically focus on narrow benchmark suites that favor one layout over another. Studies using only CNN workloads [7, 20] miss BP’s advantages in control-flow-intensive code, while those focusing on traditional CPU benchmarks [2] overlook BS’s efficiency for low-precision AI inference. The recent PIMBench suite [33] provides diverse workloads but lacks layout-aware analysis. This fragmented evaluation landscape prevents architects from understanding the true performance trade-offs. Third, architects lack principled heuristics for layout choice. Questions like "should a PIM accelerator for edge AI use bit-serial?" or "When does the overhead of supporting both layouts, as in RecNMP [19], pay off?" remain unanswered, leading to ad-hoc, suboptimal design choices.

This paper presents the first systematic, workload-driven characterization of bit-parallel versus bit-serial data layouts in PIM architectures. Rather than advocating for one layout over another, we demonstrate that **optimal layout selection is fundamentally workload-dependent**—and that ignoring this dependency incurs substantial performance penalties. Our key insight is that different computational patterns exhibit natural affinities to different layouts. We identify six fundamental challenges inherent to the BS data layout that lead to significant inefficiencies in common computational scenarios and demonstrate how adopting a Bit-Parallel (BP) data layout naturally and efficiently resolves them. We then distill these specific issues into four architectural root causes, providing a principled foundation for a more workload-driven approach to PIM design.

To quantify these effects, we base our study on SRAM-based PIM as a representative substrate and develop cycle-accurate models of both BP and BS architectures under equal-area constraints, ensuring a fair comparison within identical silicon budgets. Unlike prior work that models only computation [3], we explicitly account for bit-transposition overhead in BS designs, a cost often ignored but crucial for fair comparison [16]. Our evaluation employs a carefully curated two-tier benchmark suite: (1) microbenchmarks from MIMDRAM [29] that isolate specific computational patterns, and (2) full applications from PIMBench [33] that represent real-world workloads spanning AI inference, graph processing, and data analytics.

This paper makes the following contributions:

- **First Principled Characterization**: We provide the first systematic analysis of how bit-parallel and bit-serial data layouts in PIM architectures differ across multiple dimensions such as compute granularity, storage efficiency, control flow handling, and mixed-precision support.
- **Quantitative Performance Analysis**: Through cycle-accurate modeling of iso-area BP and BS designs, we quantify performance differences across diverse workloads, revealing up to 14× variations between static layouts. Furthermore, hybrid execution that dynamically switches between BP and BS achieves up to 2.66× speedup over the best static choice.
- **Workload-Aware Design Framework**: We develop a comprehensive taxonomy that maps workload characteristics (parallelism degree, data precision, control complexity) to optimal layout choices, synthesizing our findings into a classification framework that guides architects in selecting BP, BS, or hybrid approaches for their target applications.

The remainder of this paper is organized as follows. Section 2 provides background on PIM architectures and data layouts. Section 3 presents our architectural analysis of

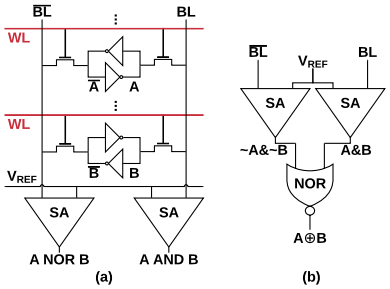


Figure 1: In-SRAM bitline operations. Simultaneous activation of two wordlines accomplishes AND/NOR operations, while an extra NOR gate facilitates XOR operation.

BP versus BS designs. Section 4 describes our experimental methodology. Section 5 presents our evaluation results across the benchmark suite. Section 6 concludes.

2. Background

Processing-in-Memory (PIM) has emerged as a compelling solution to the data movement bottleneck that characterizes modern computing. The PIM architectural landscape is broadly divided into two main categories: Processing-Near-Memory (PNM) and Processing-Using-Memory (PUM). PNM architectures integrate computational logic onto the memory controller or into the logic layer of 3D-stacked memory devices, as seen in commercial systems like Samsung's HBM-PIM [22] and UPMEM [15]. While effective at reducing data movement to and from the host CPU, PNM still treats the memory array as a passive storage unit. In contrast, PUM architectures leverage the physical properties of the memory array itself to perform massively parallel, bit-level computations directly where data resides [9, 31]. This approach offers the highest potential for parallelism and energy efficiency. Our work focuses on PUM, as the fundamental choice of data layout within the memory array—a choice with profound architectural implications—is most critical and impactful in this context.

2.1. In-SRAM Computing Primitives

PUM architectures have been built on various memory technologies, including DRAM [31] and NVM [9]. For clarity, we use SRAM-based PUM as a representative example to illustrate the fundamental computational primitives. In these systems, logic operations are executed by exploiting the intrinsic analog behavior of the memory cell array. This technique, often called in-SRAM computing, capitalizes on activating multiple wordlines simultaneously instead of just one [18]. As shown in Figure 1, this causes the SRAM cells on the selected rows to jointly discharge the bitlines (BL and $\overline{\text{BL}}$). The resulting voltage, when measured by a sense amplifier (SA), corresponds to a bit-wise logical operation across the data stored in the activated rows.

For example, an element-wise AND is realized if the SA detects a high voltage on the BL, which only occurs if all participating cells store a '1'. A NOR operation

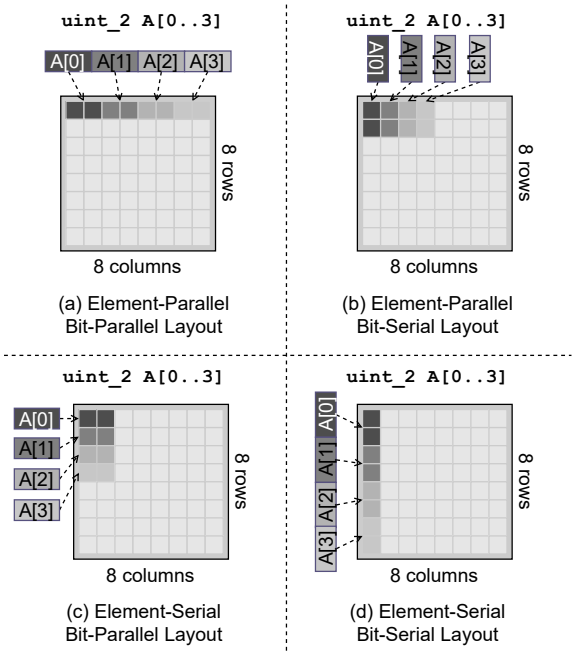


Figure 2: The four hierarchical data layout schemes resulting from combining bit-level (BP, BS) and vector-level (EP, ES) organization. (a) EP-BP: Ideal for inter-vector operations on wide data. (b) EP-BS: Maximizes parallelism for inter-vector operations. (c) ES-BP: Efficiently buffers a single vector for intra-vector operations. (d) ES-BS: Prone to row overflow for all but the simplest intra-vector tasks.

is similarly realized on the complementary bitline ($\overline{\text{BL}}$). By combining these native AND/NOR capabilities, more complex operations like XOR can also be constructed, as shown in Figure 1(b). This massively parallel computation occurs simultaneously on all columns of the array, forming the basis of PUM's efficiency. Several research efforts have built upon these primitives. For instance, Compute Cache [1] modifies the SA to support a richer set of logic operations, while Cache Automaton [36] uses a sense-amplifier cycling mechanism to accelerate specific computations. Understanding these computational primitives is key to appreciating the profound impact of data layout, which we discuss next.

2.2. Hierarchical Data Layouts

At the core of PUM design, the data layout decision bifurcates into two primary strategies for organizing data words: **Bit-Parallel (BP)** and **Bit-Serial (BS)**. This choice dictates how bits of a single word are mapped to the 2D memory grid. Orthogonal to this is the vector-level organization, which determines how multiple data elements are arranged for processing—either in **Element-Parallel (EP)** or **Element-Serial (ES)**. The interplay of these choices yields four distinct layout strategies, shown in Figure 2.

In **Bit-Parallel (BP)** layouts, an entire data word is stored horizontally across multiple columns (Figures 2a and 2c). This design, which facilitates word-level operations, has been adopted by several PUM systems such as Compute Cache [1], Ambit [31], Sealer [41], BP-NTT [40] and Inhale [42]. The BP approach is effective for both traditional SIMD-style inter-vector operations (EP-BP) and for creating efficient intra-vector scratchpads (ES-BP).

In **Bit-Serial (BS)** layouts, an N-bit word is stored vertically down a single column (Figures 2b and 2d). This approach has been the predominant strategy in PUM research, maximizing the number of parallel processing elements. It is the foundation for systems like Neural Cache [12], Duality Cache [14], SIMDRAM [16], MIMDRAM [29], and Infinity Stream [38]. While highly effective for massively parallel inter-vector operations (EP-BS), it is often impractical for storing vectors of even moderate length (ES-BS) due to the limited physical depth of memory columns, a problem we term *row overflow*.

Scope beyond SRAM.. Although our analysis and models instantiate SRAM arrays, the core layout trade-offs (parallelism versus density, vertical storage pressure, lockstep control) are technology-agnostic and often intensify in non-volatile memories due to higher write latency and limited endurance. Therefore, the workload-driven conclusions and the benefits of hybrid layout support are expected to generalise to ReRAM/PCM PUM fabrics as well.

Despite the prevalence of the BS paradigm, recent works like Proteus [28] and the PIMeval framework [33] have demonstrated the growing relevance of BP for specific workloads, particularly those requiring dynamic precision or complex intra-vector operations. However, the fundamental trade-offs between BP and BS architectures have not been systematically analyzed. This paper provides the first direct, comprehensive comparison of these two competing PUM design philosophies, aiming to guide future architects in navigating this critical design choice.

3. Architectural Principles: Challenges and Solutions

The Bit-Serial (BS) paradigm is foundational to many influential PUM systems, including Neural Cache [12], Duality Cache [14], SIMDRAM [16], MIMDRAM [29], and Infinity Stream [38], making it the de facto standard for achieving fine-grained parallelism. Despite its widespread adoption, our analysis reveals six fundamental challenges inherent to the BS data layout that lead to significant inefficiencies in common computational scenarios. This section systematically details these challenges and demonstrates how adopting a Bit-Parallel (BP) data layout naturally and efficiently resolves them. We then distill these specific issues into four architectural root causes, providing a principled foundation for a more nuanced, workload-driven approach to PIM design.

3.1. Analysis Framework and Assumptions

To provide a concrete and rigorous comparison, we ground our analysis in a set of common architectural parameters and cycle-accurate performance models for both BP and BS execution.

Architectural Parameters. We assume a conventional PIM system composed of SRAM arrays enabled for in-situ computing. The key parameters, representative of typical designs [1, 31], are a physical dimension of 128 rows (`array_rows`) and 512 columns (`array_columns`) per array, as detailed in Table 1.

TABLE 1: Architectural Parameters for Analysis.

Parameter	Value
`array_rows`	128
`array_columns`	512

Performance Models. The total latency of a kernel is the sum of load, compute, and readout cycles. While our architectural analysis in this section focuses on the compute cycles to isolate fundamental operational differences, our complete model accounts for all overheads including bit-transposition costs for BS designs and data layout conversion overheads when switching between BP and BS representations—critical factors often overlooked in prior work. Table 2 defines the cycle costs for primitive compute operations. BP operates on word-level data, where logical operations (AND, OR, NOT, XOR) and addition are single-cycle [21]. BS operates bit-by-bit, leveraging a 1-cycle hardware full adder for arithmetic and achieving zero-cost shifts by simply accessing adjacent rows [12]. A critical, costly difference for BS is the lack of a hardware multiplexer (MUX); any conditional logic must be synthesized from four primitive gates, incurring a 4-cycle penalty per bit.
Silicon evidence for single-cycle 32-bit addition. Our assumption that a 32-bit word-level add completes in one cycle is supported by measured silicon in SRAM compute arrays (e.g., 6T-bitcell datapaths operating at multi-GHz frequencies) and by published BP-IMC designs reporting one-cycle ADD latency [21]. We therefore treat word-level logical/arithmetic primitives as single-cycle operations in the BP model.

TABLE 2: Compute Cycle Costs for BP and BS Primitives.

Bit-Parallel (BP) Mode		Bit-Serial (BS) Mode	
Operation	**Cycles**	**Operation**	**Cycles**
Logic (N-bit)	1	1-bit Add/Sub	1
ADD (N-bit)	1	Shift	0
SUB (N-bit)	2	1-bit MUX	4
MULT (N-bit)	N+2		
SHIFT (k bits)	k		

3.2. Fundamental Challenges of Bit-Serial Data Layout

Challenge 1 - Severe Underutilization with Limited Parallelism: While PIM is often associated with massively

parallel workloads like processing an entire image, many critical, real-time applications involve operating on small, sparse subsets of data. Consider an augmented reality system that must highlight 16 distinct points of interest on a high-resolution video stream. This requires adding a color-correction vector to the 16 corresponding pixel vectors, which is a classic vector addition task. The system must process this small batch of operations with minimal latency to maintain real-time performance, presenting a low degree of parallelism (DoP) workload to the PIM accelerator.

```
// A[]: original 32-bit pixel color vectors
// B[]: 32-bit color correction vectors
// C[]: resulting pixel color vectors
for (int i = 0; i < 16; ++i) {
    C[i] = A[i] + B[i];
}
```

Listing 1: A vector addition kernel with a DoP of 16, analogous to applying color correction to 16 pixels.

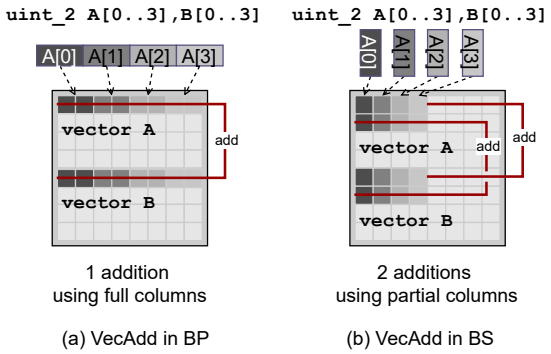


Figure 3: Contrasting a 4-element vector addition in (a) Bit-Parallel (BP) and (b) Bit-Serial (BS) layouts. BP configures the array into four wide PEs, utilizing the full array width. BS uses only four 1-bit columns, leaving most of the hardware idle.

Analysis. The kernel in Listing 1 has a DoP of 16. As illustrated conceptually in Figure 3b, a BS architecture must dedicate a separate column to each of the 16 parallel operations. This activates only 16 of the 512 available 1-bit PEs, resulting in a dismal 16/512 $\approx$ 3.1% resource utilization. The **BP solution** (Figure 3a), however, perfectly matches the hardware to the workload. By configuring the array into 16 PEs, each 32 bits wide for the pixel data, the kernel utilizes all $16 \times 32 = 512$ columns, achieving 100% resource utilization.

Challenge 2 - Inefficient On-Chip Buffering and Row Overflow: Finite Impulse Response (FIR) filters are a cornerstone of digital signal processing, widely used in applications from audio equalizers to image sharpening and wireless communications. The core computation of an N-tap FIR filter is a convolution: it calculates a weighted sum of the ‘N‘ most recent input samples. This inherently requires maintaining a sliding window of past inputs (‘x[n]‘, ‘x[n-1]‘, etc.) and the filter's coefficients (‘b0‘, ‘b1‘, etc.) in on-chip memory for fast access. The efficiency of the filtering process thus hinges on using the PIM array as a high-bandwidth, software-managed scratchpad.

```
// state[]: holds 4 previous input samples (the
    sliding window)
// coeffs[]: holds 4 filter coefficients
// new_sample: the next input from a data stream

// 1. Update state: shift old samples
state[3] = state[2];
state[2] = state[1];
state[1] = state[0];
state[0] = new_sample;

// 2. Compute output: convolution
output = coeffs[0]*state[0] + coeffs[1]*state[1] +
         coeffs[2]*state[2] + coeffs[3]*state[3];
```

Listing 2: Core loop of a 4-tap FIR filter, showing the state update process.

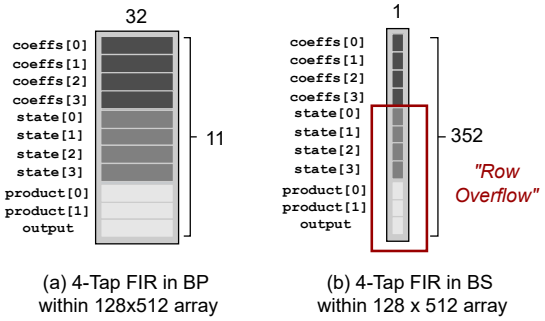


Figure 4: Physical data layout for a 4-tap FIR filter. (a) In the BP layout, all required variables—coefficients, state, intermediate products, and the final output—are stored in separate rows, comfortably fitting within the array. (b) The BS layout attempts to store all these variables vertically, causing a massive row overflow.

Analysis. The FIR filter logic in Listing 2 requires not only the 4 ‘state‘ samples and 4 ‘coeffs‘ to be resident, but also space for intermediate products and the final accumulator (at least 11 word-level variables in total, as shown in Figure 4). In a BS layout (Figure 4b), storing these 32-bit words vertically would require $11 \times 32 = 352$ rows. This far exceeds the physical `array_rows` of 128, causing a severe **row overflow** and forcing costly data eviction. The **BP solution** (Figure 4a) is natural: each 32-bit word occupies a separate row, fitting comfortably within the 128 available rows.

Challenge 3 - Inefficient Intra-Vector Operations: Intra-vector permutations are a critical performance bottleneck in modern cryptography. The Keccak hash function, the winner of the SHA-3 competition, is a prime example. Its π stage, a core part of its sponge construction, applies a fixed, non-trivial permutation to the elements of its internal state vector in every round. The efficiency of this data-

agnostic shuffling operation is paramount to the overall performance of the algorithm.

```
// state[]: a vector of 25 64-bit words,
    representing the Keccak state
// next_state[]: a temporary vector to hold the
    permuted state

// The Pi permutation shuffles elements according
    to a fixed pattern.
next_state[0] = state[0];
next_state[1] = state[15];
next_state[2] = state[5];
next_state[3] = state[20];
// ... more permutations ...
next_state[22] = state[4];
next_state[23] = state[19];
next_state[24] = state[9];
```

Listing 3: Conceptual permutation (π stage) in Keccak.

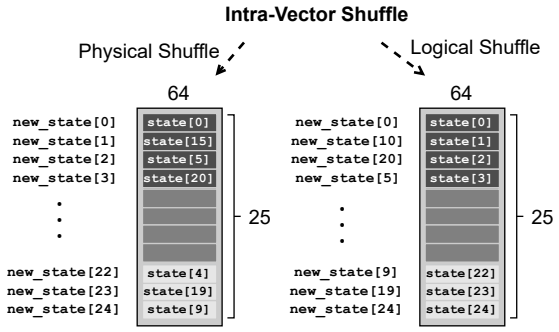
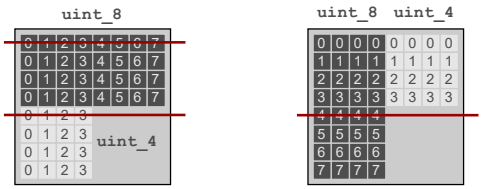


Figure 5: Contrasting two permutation mechanisms. Left: a physical shuffle requires explicit data movement between memory locations, incurring multiple read-write cycles. Right: a logical shuffle achieves the same permutation through zero-cost address remapping, native to BP layouts with Element-Serial organization.

Analysis. The operational difference is stark, as illustrated in Figure 5. The key distinction lies in how permutations are implemented: a **physical shuffle** requires actual data movement between memory locations, while a **logical shuffle** achieves the same result through address remapping without moving any data. The left side of Figure 5 shows a physical shuffle where elements must be explicitly copied from their source locations to new destinations—an operation requiring multiple read-write cycles. The right side demonstrates a logical shuffle where the permutation is achieved by simply updating the mapping between logical indices and physical addresses, incurring zero data movement cost.

In the context of Keccak's π permutation, a BS architecture cannot use an Element-Serial (ES) layout due to massive row overflow (25 64-bit elements would require 1,600 rows). It is forced into an Element-Parallel (EP-BS) layout, where elements reside in different columns. Permuting them requires the costly physical shuffle shown on the left—a sequence of explicit inter-column data transfers. In contrast, a BP architecture using an ES-BP layout naturally performs the π permutation via the logical shuffle shown on the right, completing the entire operation in zero cycles through address remapping.

Challenge 4 - Inflexible Mixed-Precision Execution: Mixed-precision computation is a cornerstone of efficient deep learning, where models are often quantized to use a mix of 8-bit, 4-bit, and even lower-precision data types to reduce memory and compute costs.



(a) Mixed Precision in BP (b) Mixed Precision in BS

Figure 6: Mixed-precision execution. (a) In a BP layout, 8-bit and 4-bit data are processed by independent, word-level PEs. (b) In a BS layout, the bit-level synchronous control creates a conflict: processing bit 4 is valid for the 8-bit data but is out of bounds for the 4-bit data.

Analysis. The flexibility of the BP layout is shown in Figure 6(a). Here, 8-bit and 4-bit vectors can be laid out in an ES-BP format. The PIM array is configured into independent, word-level PEs that process their respective data types without conflict. In stark contrast, the BS layout (Figure 6(b)) faces a fundamental synchronization problem. To process different vectors in parallel, an EP-BS layout is used. When the global controller issues the command to process bit 4, it is a valid operation for the 8-bit data but results in an **out-of-bitwidth error** for the 4-bit data. This conflict makes true parallel execution of mixed-precision data impractical in BS architectures, forcing inefficient workarounds like padding all data to the largest precision, which negates the benefits of using smaller data types.

Challenge 5 - Impractical Handling of Complex Control Flow: Branching instructions are ubiquitous in general-purpose code. However, PIM architectures, much like GPUs, are ill-suited for traditional control flow where different parallel lanes might take different paths. To avoid breaking parallelism, PIM systems handle branches using **predicated execution**: both sides of a branch are computed unconditionally, and the correct result is then selected using a flag or a bitmask derived from the branch condition. This transforms a control hazard into a data storage requirement, a trade-off that has profound implications for the underlying data layout.

```
// Original code: if (A > B) { C = D * E; } else {
    C = F + G; }

// 1. Compute condition (A > B) and create bitmask
// This is done by checking the sign bit of (B - A
    ).
uint32_t sub = B - A;
```

```
// Arithmetic shift right broadcasts the sign bit. If A > B,
// B-A is negative, its sign bit is 1, so the mask becomes all 1s.
uint32_t mask = sub >> 31; // For 32-bit integers

// 2. Compute both branches unconditionally
uint32_t res_true = D * E;
uint32_t res_false = F + G;

// 3. Select the correct result using the mask
C = (res_true & mask) | (res_false & ~mask);
```

Listing 4: Predicated execution of a conditional statement.

Analysis. The predicated execution in Listing 4 requires that all operands (A, B, D, E, F, G) and intermediate results (`sub`, `mask`, `res_true`, `res_false`) be stored simultaneously. This amounts to at least 10 word-level variables that must be resident in the PIM array. As demonstrated by the FIR filter example (Challenge 2), this storage requirement is impractical for a BS layout. Storing these 32-bit words vertically would require $10 \times 32 = 320$ rows, causing a severe **row overflow** of our 128-row array. The **BP solution**, by contrast, easily accommodates this by storing each variable in a separate row, making predicated execution a viable and efficient strategy.

Challenge 6 - High Inherent Latency for Word-Level Operations: For latency-critical applications such as real-time vehicle control or interactive database queries, the absolute time to complete an operation is more important than aggregate throughput.

TABLE 3: Compute Cycle Latency for Common 32-bit Kernels.

Microkernel	BP Cycles	BS Cycles
Vector Addition	1	32
Vector Multiplication	34	1024
MIN / MAX	36	192
If-Then-Else	7	97

Analysis. The high latency of BS is not an isolated issue but a fundamental characteristic. As shown in Table 3, for a 32-bit addition, the BP solution is 32× faster. This latency gap explodes for more complex operations. A 32-bit multiplication in BP takes only 34 cycles, while the bit-serial equivalent requires over 1000 cycles. Even for conditional logic ('If-Then-Else'), BP maintains a significant advantage. For applications where every cycle counts, this dramatic difference in computational latency makes BP the clearly superior architecture.

3.3. Synthesis: Architectural Root Causes of Bit-Serial's Inefficiencies

The six challenges detailed previously are not isolated flaws; they are symptoms of four fundamental and intertwined architectural trade-offs inherent to the bit-serial design philosophy:

- **Granularity Mismatch.** The rigid, fine-grained parallelism of the BS architecture is ill-suited for workloads with limited or variable degree of parallelism. This mismatch directly causes the severe resource underutilization demonstrated in **Challenge 1**, where a vast majority of the hardware remains idle.
- **Vertical Storage Bottleneck.** Storing N-bit words vertically down a column of finite depth is a core tenet of the BS model, but it creates a critical storage bottleneck. This single design choice is the primary source of inefficiency across three distinct scenarios: it renders on-chip buffering for algorithms like FIR filters impractical (**Challenge 2**); it precludes efficient intra-vector operations such as cryptographic shuffles (**Challenge 3**); and it makes predicated execution for complex control flow infeasible due to excessive storage demands (**Challenge 5**). This "row overflow" problem is arguably the most pervasive architectural weakness of the BS paradigm.
- **Lockstep Control Conflict.** The reliance on a single, global control signal broadcast to all 1-bit PEs creates a fundamental conflict when processing heterogeneous data types. This was demonstrated in the mixed-precision case (**Challenge 4**), where a command valid for one data width is out-of-bounds for another, thus negating the possibility of true parallel execution.
- **Inherent Computational Latency.** By its very definition, bit-serial processing imposes a latency of at least N cycles for any N-bit operation. This results in high latency for all word-level tasks, a weakness starkly quantified by the data in **Challenge 6**, where the performance gap between BS and BP can be orders of magnitude for common operations.

In conclusion, these challenges demonstrate that the implicit "one-size-fits-all" assumption favoring bit-serial layouts is fundamentally flawed. Bit-parallel architectures are not merely an alternative but offer robust solutions to these widespread challenges. The optimal choice of data layout is, therefore, not fixed but is strongly dependent on workload characteristics, demanding a more nuanced, workload-aware approach to PIM system design.

On increasing array rows.. Some BS limitations (e.g., small working scratchpads) can be alleviated by increasing the number of physical rows. However, this exacerbates array area, wire delay, and sensing energy, and does not address the root causes we identify—vertical storage bottleneck, lockstep control conflict, and inherent bit-serial latency—so the qualitative trade-offs remain.

4. Evaluation Methodology

To provide the first systematic, workload-driven comparison of bit-parallel (BP) and bit-serial (BS) data layouts, we establish a rigorous analytical framework. This framework is grounded in detailed, cycle-accurate performance models of two iso-area Processing-in-Memory

(PIM) architectures—one implementing BP execution and the other implementing BS execution—ensuring a fair comparison under identical silicon resource constraints. Our methodology leverages the architectural cost models developed in Section 3 to precisely quantify performance across a diverse set of computational patterns, foregoing traditional hardware simulation for a more direct, model-based analysis.

4.1. Architectural Model

Our study targets a single 128 row × 512 column Computing SRAM Array (CSA) that can execute in two mutually exclusive modes: *bit–parallel* (BP) and *bit–serial* (BS). The physical memory array, sensed by 6T bitcells identical to Lee *et al.* [21], is shared; only the peripheral logic differs, ensuring an *iso–area* comparison.

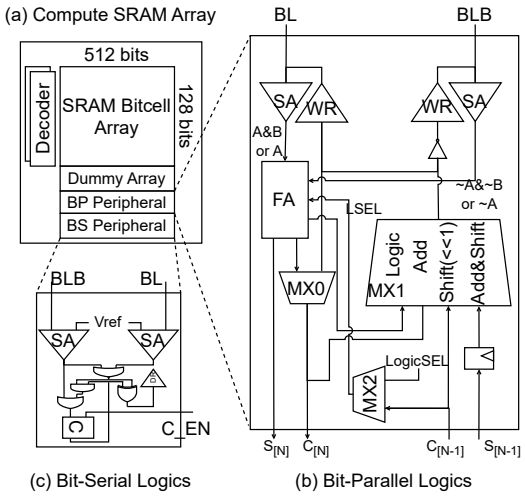


Figure 7: Compute SRAM array *(a)* with dual peripherals: a word–level BP datapath *(b)* and a 1–bit BS datapath *(c)* sharing the same 128 × 512 cell core. Mode selection is performed by a lightweight mux on each column sense line.

Bit–Parallel (BP) Peripheral. The BP peripheral aggregates neighbouring bitlines into word–level Processing Elements (PEs). The array can be *reconfigured at run time* to any word width from 2 to 32 bits, trading off parallelism for precision. Word–level logic and arithmetic instructions execute in a single cycle; multi–cycle operations such as multiplication follow the cycle costs in Table 2. Logical permutations (e.g., Keccak π) are realised as zero–cycle address remaps.

Bit–Serial (BS) Peripheral. Switching to BS mode disables the word aggregators and exposes each of the 512 columns as an independent 1–bit PE, akin to Neural Cache [12]. Arithmetic is performed bit by bit with a 1–cycle full adder, and shifts are free by virtue of adjacent rows. Conditional logic is synthesised from primitives, incurring a 4–cycle `MUX` penalty per bit. We model the overhead of intra–vector *physical* shuffles required when data must move across columns.

Because both peripherals attach to the same CSA core, their silicon footprints differ only in peripheral circuits; we conservatively allocate equal metal layers so that total die area remains constant, upholding the iso–area assumption used throughout the evaluation.

Iso–area justification.. Across SRAM–PIM implementations, published area breakdowns consistently show that the cell array dominates die area (often $> 90\%$), while peripheral logic contributes a single–digit percentage. This trend holds for both word–level (BP) and 1–bit (BS) datapaths [1, 12, 21]. Consequently, allocating equal metal and periphery budgets to BP and BS yields a fair, iso–area comparison.

On–Chip Transpose Unit. Many applications benefit from switching between BP and BS representations. We model an on–chip transpose unit attached to the column sense lines that performs bit/word transposition in hardware. The *core* transpose is a single cycle at GHz–class speeds, consistent with prior bitline shuffle hardware [16]. End–to–end transpose latency is dominated by array read and write to feed and drain the unit. For an M–row logical object in BP and N rows in BS, the total cost is $\text{read}(M)+1+\text{write}(N)$ cycles in the BP→BS direction and $\text{read}(N)+1+\text{write}(M)$ in the reverse direction. These costs match the per–round accounting used in our AES study (Section 5.4).

4.2. Model Validation

Our primitive cycle costs are grounded in measured or reported silicon data from SRAM compute peripheries and in–array logic [12, 16, 21]. We validate our analytical totals against two public benchmark suites, MIMDRAM [29] (micro–kernels) and PIMBench [33] (applications). The relative trends we report agree with the layout–specific results produced by the PIMeval framework [33], providing confidence in the model's fidelity.

4.3. Benchmark Suite

To comprehensively evaluate the trade-offs between BP and BS, we employ a curated two-tier benchmark suite designed to expose the layout affinities of diverse computational patterns.

4.3.1. Tier 1: Microbenchmarks. To isolate the performance of fundamental computational patterns, we select a broad set of microbenchmarks inspired by the MIMDRAM [29] suite. These kernels include vector arithmetic, logical and comparison operations, control flow primitives, and data organization patterns like reductions. These microbenchmarks allow us to directly measure and validate the performance characteristics and latency costs detailed in our architectural models.

4.3.2. Tier 2: Application Benchmarks. To assess performance on real-world workloads, we draw from the

PIMBench [33] suite, which covers a wide range of application domains. Our selection includes representative kernels from Deep Learning (VGG), Cryptography (Keccak), Signal Processing (FIR filters), and Data Analytics (database-style queries). By analyzing these full applications, we can demonstrate how the performance of individual computational patterns aggregates at the workload level, thereby providing the evidence needed to develop our final layout selection guidelines.

5. Evaluation

5.1. Evaluation Goals

Section 3 distilled *six concrete challenges* of the canonical bit-serial (BS) layout and traced them back to *four architectural root causes*: (*i*) granularity mismatch, (*ii*) vertical storage bottleneck, (*iii*) lock-step control conflict, and (*iv*) inherent computational latency. Our evaluation therefore aims to determine *when and why* each data layout succeeds or fails with respect to these root causes. We use a two-tier benchmark suite that couples micro-kernels and full applications.

We address the following three questions:

1) **Q1 — Layout → Kernel Mapping:** For the diverse micro-kernels that dominate modern workloads—arithmetic, logical, reduction, and control-intensive operations—how do BP and BS compare in compute latency, and which root cause is the primary limiter for each kernel category?
2) **Q2 — Density vs. Latency Trade-off:** As data sets grow beyond a single array's capacity, batching becomes mandatory. How does the tension between BP's lower storage density and BS's higher per-operation latency influence end-to-end execution time, and where is the inflection point where batching overhead neutralises BP's compute advantage?
3) **Q3 — Application-Level Impact and Adaptivity:** When kernels with conflicting layout preferences coexist within one application (e.g., AES combining table look-ups and matrix multiplications), what is the net performance of a *static* BP or BS choice? Can a *hybrid* strategy that switches layouts at phase boundaries offset individual weaknesses despite transposition overheads?

Answering these questions provides workload-aware guidelines for selecting—or dynamically combining—data layouts, validating our claim that no single layout is universally optimal across the PIM design space.

5.2. Experimental Setup

Hardware Assumptions. All results are obtained with the iso-area bit-parallel (BP) and bit-serial (BS) array organizations defined in Section 4. Unless stated otherwise, we use the baseline configuration of 128 rows and 512 columns per array (Table 1) and the cycle-level cost model of Table 2. Both layouts share the same total silicon footprint; differences in performance stem solely from data placement and execution style.

Benchmark Suite. We directly reuse the two-tier benchmark suite introduced in Section 4.3. Tier 1 comprises 18 micro-kernels that exercise arithmetic, logical, reduction, and control-flow patterns, evaluated with two input sizes: *small* (1 024 elements) and *large* (65 536 elements). Tier 2 contains 22 full applications spanning vision, learning, analytics, and cryptography; each uses widely adopted dataset dimensions (e.g., CIFAR-10 for VGG-16, 1 M points for K-means) and identical inputs across layouts.

Methodology. For each layout we compute total latency as cycles = load + compute + readout, where *compute* cycles come directly from the primitives in Table 2. Load and readout costs scale linearly with the number of active rows and therefore reflect batching overhead when BP overflows the array. End-to-end runtimes are normalised to a 1 GHz array clock and reported in cycles for maximum portability.

5.3. Tier–1: Microbenchmark Analysis

Our microbenchmark analysis proceeds in two steps, using precise data from Tables 5 and 4. First, we examine performance at a fixed scale (1024 elements) to understand core computational trade-offs. Second, we analyze sensitivity to workload size to reveal the critical impact of data layout density and batching.

Performance at Fixed Scale.. Table 5 provides a cycle-level breakdown that reveals a wide performance spectrum, directly tracing back to the architectural challenges.

The *Arithmetic* cluster highlights the raw efficiency of BP's word-level datapath. For basic operations like `Vector Add`, BP's 'Compute' time is just 1-2 cycles, while BS requires 16, a direct result of its **Inherent Latency (Challenge 6)**. The gap widens for `MULTU`, where BP's 18-cycle hardware is over 14× faster than the 256-cycle BS shift-and-add implementation.

The *Logical / Bit-manipulation* cluster reveals a more nuanced picture. For bit-centric tasks like `bitcount`, the **Granularity Mismatch (Challenge 1)** makes BP less efficient; its total latency (185 cycles) is higher than the more natural serial summation in BS (128 cycles). Conversely, the native serial 'Reduction' in BS is highly optimized and slightly outperforms the BP tree-based version.

The *Control / Predicate* cluster demonstrates BP's performance advantage for conditional operations. Kernels such as `abs` and `if-then-else` require conditional logic that BS must synthesize through multiplexers, increasing compute cycles from 7 (BP) to 49 (BS) for `if-then-else`—a consequence of the **Lock-step Control Conflict (Challenge 4)**. These kernels also exhibit the **Vertical Storage Bottleneck (Challenge 2)**: the `if-then-else` kernel's 10 live variables require 52 rows in BS versus 5 in BP, indicating potential overflow issues in complex predicated code.

Sensitivity to Workload Size.. The fixed-scale analysis does not capture the full picture. End-to-end performance is critically sensitive to workload size due to BP's lower

storage density. Table 4 uses vector addition to illustrate this "batching effect" with precise figures.

TABLE 4: Vector addition latency as a function of workload size. BP batches increase once the working set exceeds 16K elements, neutralising its compute advantage. Speedup is BS/BP.

Elements	BP Batches	BP Cycles	BS Cycles	Speedup
1K	1	97	112	1.15×
4K	1	385	400	1.04×
16K	1	1,537	1,552	1.01×
64K	4	6,148	6,160	1.00×
256K	16	24,592	24,592	1.00×

For 1K-element workloads, BP achieves a 1.15× speedup over BS. This advantage decreases to 1.01× at 16K elements—BP's single-batch capacity—as data movement costs (load/readout) become significant. At 64K elements, BP requires 4 sequential batches, and the overhead of repeated data loading eliminates its computational advantage, resulting in performance parity with single-batch BS execution. This demonstrates the fundamental trade-off between BP's low latency (Challenge 6) and its lower storage density (Challenge 2).

Synthesis.. The results demonstrate that optimal layout selection depends on both *kernel characteristics* and *workload size*. At small scales, kernel-specific properties determine performance: BP achieves up to 14× speedup for arithmetic operations (`MULTU`), while BS performs 1.4× better for bit-level operations (`bitcount`). As workload size increases, BP's lower storage density necessitates batching, which diminishes or eliminates its computational advantage. These findings support workload-aware PIM architectures over static layout choices.

Implication—batching neutralizes advantages.. The measurements confirm that layout performance depends on both kernel characteristics and workload size. BP's latency advantage applies only when the working set fits within a single array batch. When batching is required, data movement costs dominate execution time, causing both layouts to achieve similar performance. This size sensitivity explains the reduced speedups observed in application-level benchmarks (Section 5.4) compared to micro-kernels, reinforcing the need for workload-aware PIM architectures.

5.4. Tier–2: Application Benchmark Results

Micro-kernels provide architectural intuition, but real programs combine many kernel types and exhibit diverse working-set footprints. We therefore profile 22 end-to-end applications drawn from vision, learning, analytics, and cryptography. The performance metric is total kernel cycles under the BP and BS layouts, using the analytical framework of Section 5.2. To expose the abundant thread- and data-level parallelism of these real-world workloads, we assume a system with **512 parallel arrays**, scaling the core architecture of Section 4 accordingly. Table 6 groups the workloads by their observed speedup and dominant limiting factor.

Case study 1: VGG-13 inference (resource utilization).. VGG-13 inference [34] demonstrates how resource utilization varies across convolutional layers as feature map dimensions decrease. Figure 8 quantifies this effect by comparing each layer's output size (assuming 16-bit elements and 3×3 kernel reuse) against the PIM's maximum parallelism of 262,144 bits.

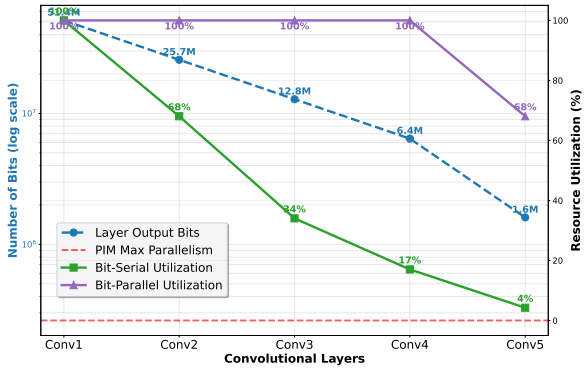


Figure 8: VGG-13 layer output size and PIM resource utilization. Left axis (log scale) shows layer output bits versus PIM capacity. Right axis shows the resulting resource utilization for BP and BS.

The utilization patterns reveal fundamental architectural differences:

- **Early layers (Conv1-Conv3):** Both layouts achieve 100% utilization, as the feature maps exceed the PIM's processing capacity. The performance difference stems solely from BP's lower computational latency (Challenge 6).
- **Later layers (Conv4-Conv5):** BS utilization decreases to 17% (Conv4) and 4% (Conv5) due to insufficient parallelism—the 16-bit element count cannot saturate 262,144 bit-serial units. BP maintains higher utilization (100% for Conv4, 68% for Conv5) through its word-level granularity, which better matches the reduced workload dimensions.

These measurements demonstrate that BS suffers from the **Granularity Mismatch (Challenge 1)** in later network layers. While early layers provide sufficient parallelism for both layouts, the progressive reduction in feature map sizes leaves BS compute resources increasingly underutilized. This quantitative evidence contradicts the assumption that BS universally provides superior efficiency in neural network inference.

Case study 2: AES-128 encryption (hybrid wins).. AES-128 encryption combines four distinct operations with strongly conflicting layout preferences, making it an ideal showcase for hybrid execution. Table 7 quantifies the dramatic performance disparities across AES stages.

The performance disparity stems from fundamental architectural mismatches:

- **SubBytes:** The S-box lookup is the most expensive operation in BP, requiring complex GF(2^8) inversion

TABLE 5: Cycle breakdown of advanced micro-kernels. Rows/Elem and Cols/Elem denote physical footprint per element in the array. All values are for 32-bit data unless noted.

Kernel	Variant	Mode	Rows/Elem	Cols/Elem	Load	Compute	Readout	Total	Challenge
Arithmetic kernels									
Vector Add	Standard	BP	~3	16	64	1	32	97	6
		BS	49	1	64	16	32	112	6
Vector Sub	Standard	BP	~3	16	64	2	32	98	6
		BS	49	1	64	16	32	112	6
MULTU	HW	BP	~4	16	128	18	64	210	6
	Shift+Add	BS	64	1	64	256	64	384	6
MULTU Const	HW	BP	~3	16	128	18	64	210	6
	Shift+Add	BS	48	1	64	256	64	384	6
DIVU	Restoring	BP	4	16	64	640	32	736	6
	Restoring	BS	64	1	64	1280	32	1376	6
MIN	Shift Mask	BP	~5	16	64	21	32	117	6
	Iter. Comp.	BS	50	1	64	96	32	192	6
MAX	Shift Mask	BP	~5	16	64	21	32	117	6
	Iter. Comp.	BS	50	1	64	96	32	192	6
Logical / Bit–manipulation kernels									
Reduction	Tree	BP	2	16	32	19	16	67	6
	Native	BS	17	1	32	16	16	64	6
bitcount	D&C	BP	3	16	128	25	32	185	1
	Summation	BS	26	1	32	80	16	128	1
bitweave	1b Logic	BP	53	1024	96	225	2	323	1
	2b Logic	BS	74	512	64	434	2	500	1
	4b Logic	BS	116	256	48	852	2	902	1
Control / Predicate kernels									
abs	Shift Mask	BP	3	16	32	18	32	82	4
	Serialised	BS	48	1	32	48	32	112	4
if-then-else	Mask 0-s	BP	5	16	96	7	32	135	2/6
	Synth. MUX	BS	52	1	80	49	32	161	2/6
equal	XOR+Reduce	BP	3	16	64	22	32	118	6
	Serial XOR	BS	49	1	64	33	32	129	6
ge_0	Shift	BP	1	16	32	17	16	65	6
	Sign Bit	BS	16	1	32	1	16	49	6
gt_0	Synth.	BP	3	16	32	35	32	99	6
	Serial Red.	BS	17	1	32	17	16	81	6
ReLU (8K)	Standard	BP	~2	16	512	17	512	1041	4
	Standard	BS	~17	1	512	17	512	1041	4

implemented via composite field arithmetic. This consumes approximately 98 cycles per byte, totaling 1,568 cycles for 16 bytes. BS, however, leverages the Boyar-optimized bit-sliced implementation [6] that transforms the S-box into just 115 logic gates, completing in 115 cycles—a remarkable 13.6× speedup.

- **Other stages:** AddRoundKey (XOR), ShiftRows (byte permutation), and MixColumns (GF multiplication) are naturally word-oriented operations where BP excels with its native 8-bit datapath, achieving 8× better performance than BS.

A *hybrid* execution strategy capitalizes on these complementary strengths: execute SubBytes in BS, all other operations in BP, with layout transpositions between phases. The total cost becomes:

$$\text{Per round: } 16 + 115 + 32 + 272 + 290 = 725 \text{ cycles}$$

where the 290-cycle transposition overhead[1] is amortized across the round.

Sensitivity to transpose cost. Our conclusions are robust to slower transpose hardware. If we conservatively increase the transpose cost by 10× (core latency from 1 to 10 cycles), the total AES runtime increases by only ∼2.6%, and the hybrid schedule still achieves a 2.59× speedup over the best static layout (BP). Thus, the hybrid benefit is insensitive to transpose latency within a wide, practical range.

1. The 290-cycle cost includes two transpositions: BP→BS (145 cycles) before SubBytes and BS→BP (145 cycles) after. Each 145-cycle transposition breaks down as follows: (1) reading data to the transposer: 16 cycles for BP→BS or 128 cycles for BS→BP, (2) hardware transpose operation: 1 cycle, and (3) writing back: 128 cycles for BP→BS or 16 cycles for BS→BP. The asymmetry reflects the different row counts: AES state occupies 16 rows in BP (1 bytes/row × 16 rows) but 128 rows in BS (1 bit/row × 128 bits).

TABLE 6: Classification of application benchmarks. Speedup is reported as BS/BP; values < 1 indicate BS is faster.

Category	Applications	Speedup (BS/BP)	Dominant Architectural Factor
Strong BP preference	Brightness, K-means, Keccak, FIR	1.5–3.0×	Mixed arithmetic / control (Challenges 4,6)
Moderate BP preference	VGG-13, VGG-16/19, GEMM, GEMV, Conv, Downsample	1.2–1.5×	High arithmetic intensity, limited batching (6)
Balanced	Vector-Add, AXPY, Pooling, Prefix-Sum	1.0–1.15×	Batching neutralises latency (2)
BS preference	Histogram, HDC, Bitweave-DB	0.6–0.9×	Bit-centric, full-density layouts (1)
Hybrid recommended	AES, Radix-Sort	2.66× speedup*	Phase diversity (3,4,5)

*Hybrid speedup is relative to the best static layout (BP for AES).

TABLE 7: Per-round cycle breakdown for AES-128 operations.

Operation	Bit-Parallel	Bit-Serial	Best Layout
AddRoundKey	16	128	BP (8×)
SubBytes	1,568	115	BS (13.6×)
ShiftRows	32	256	BP (8×)
MixColumns	272	2,176	BP (8×)
Total per round	1,888	2,675	-

For the complete AES-128 encryption (10 rounds):

- Pure BP: 18,624 cycles (SubBytes dominates at 83.1% of runtime)
- Pure BS: 26,750 cycles (MixColumns becomes the bottleneck)
- Hybrid: 6,994 cycles (balanced execution with transposition overhead)

This **2.66**× improvement over the best static layout (BP) demonstrates that judicious layout switching can overcome the individual weaknesses of each paradigm. The win is particularly compelling for AES because: (1) SubBytes has an extreme 13.6× performance advantage in BS due to bit-sliced optimization, (2) the transpose cost (2,900 cycles total) is only 20% of the compute savings (14,530 cycles), and (3) the regular round structure enables predictable, compiler-driven layout management.

Takeaway.. Application results corroborate the kernel analysis: (1) BP's latency advantage translates to $\sim 1.3\times$ median speedup for arithmetic-heavy workloads; (2) bit-centric tasks favour BS despite its higher per-operation latency; (3) hybrid execution can exceed either static layout when phase diversity is high. These findings motivate future PIM designs that support fast, low-energy layout transposition rather than locking the system to one paradigm.

Energy considerations.. While we defer a full energy model to extended work, measured silicon data already suggests that word-parallel SRAM-PIM can be highly energy-efficient. Reported TOPS/W for ADD in SRAM PIM shows BP-style datapaths achieving higher energy efficiency than BS-style ones in comparable technologies (e.g., ∼8.1 TOPS/W [21] vs. ∼5.3 TOPS/W [37] for ADD). Combined with our latency analysis, these figures indicate that the most energy-efficient layout is workload-dependent, and hybrid strategies that minimise time spent in an energy-inefficient layout can further reduce energy.

5.5. Synthesis and Key Takeaways

The three evaluation tiers paint a coherent picture: neither data layout unilaterally dominates; instead, each excels under specific workload conditions. Table 8 summarises the qualitative trends distilled from more than forty quantitative data points.

TABLE 8: Workload characteristics favoring each layout.

BP-friendly	BS-friendly
Word-level arithmetic (add/mul/-div)	Bit-level operations (popcount, XOR)
Conditional logic, predication	Uniform, data-independent control
Mixed precision vectors	High DoP, full utilization
Latency-critical tasks	Large working sets
Low degrees of parallelism	Logical transpositions

Hybrid data layouts as the next frontier.. The AES case study underscores that dynamic layout switching—enabled by an on–chip transpose engine—can beat the best static choice. Our analytical model shows that when transpose cost is below 2% of per–phase runtime (51 cycles in our configuration), a hybrid schedule is profitable for any application that contains at least one BS–favourable and one BP–favourable phase. This opens an attractive design space between the two historical extremes.

Future work.. Two directions merit further investigation: (1) fine -grained, row–selective transpose units that amortise cost over partial data sets; and (2) compiler analyses that automatically partition code into layout–optimal regions and insert transpose operations only when the expected gain exceeds the hardware threshold. We leave the exploration of these co–optimised hardware–software techniques to future work.

In summary, our evaluation confirms the central thesis: *data layout is a first–order design decision for PIM.* A workload–aware architecture that flexibly navigates the BP/BS continuum stands to unlock significant, otherwise untapped performance. The methodology and empirical findings presented here provide a quantitative foundation for such adaptive systems.

6. Conclusion

This paper presented the first systematic comparison of bit-serial (BS) and bit-parallel (BP) data layouts in Processing-in-Memory architectures. Through cycle-accurate modeling and evaluation across 22 applications, we identified six fundamental challenges of BS layouts and delivered three key insights: **Architecture matters.** BP achieves up to 14× lower latency than BS for arithmetic and control-heavy kernels by eliminating serial carry propagation. **Density matters more.** When workloads exceed array capacity, BP's batching overhead neutralizes

its computational advantages, enabling BS to excel on bit-centric or large-scale problems. **Flexibility wins.** Hybrid execution that switches layouts at phase boundaries outperforms static choices by up to 2.66× when transposition cost is minimal. These findings transform data layout from an implicit assumption into a first-class design parameter. Future PIM systems should support low-cost transposition hardware and workload-aware compilers that orchestrate dynamic layout decisions. By establishing that neither BP nor BS is universally optimal, this work motivates adaptive architectures that navigate the BP/BS continuum based on workload characteristics.

References

[1] S. Aga, S. Jeloka, A. Subramaniyan, S. Narayanasamy, D. Blaauw, and R. Das, "Compute caches," in *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. ieeexplore.ieee.org, pp. 481–492.

[2] J. Ahn, S. Yoo, O. Mutlu, and K. Choi, "PIM-enabled instructions: a low-overhead, locality-aware processing-in-memory architecture," vol. 43, pp. 336–348.

[3] S. Angizi, Z. He, and D. Fan, "PIMA-logic: a novel processing-in-memory architecture for highly flexible and energy-efficient logic computation," in *Proceedings of the 55th Annual Design Automation Conference*. ACM.

[4] H. Asghari-Moghaddam *et al.*, "Chameleon: Versatile and practical near-dram acceleration architecture for large memory systems," in *MICRO*, 2019.

[5] A. Boroumand, S. Ghose, Y. Kim, R. Ausavarungnirun, E. Shiu, R. Thakur, D. Kim, A. Kuusela, A. Knies, P. Ranganathan, and O. Mutlu, "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*. ACM.

[6] J. Boyar and R. Peralta, "A new combinational logic minimization technique with applications to cryptology," in *Experimental Algorithms*. Springer Berlin Heidelberg, pp. 178–189.

[7] Y.-H. Chen, J. Emer, and V. Sze, "Eyeriss: a spatial architecture for energy-efficient dataflow for convolutional neural networks," vol. 44, pp. 367–379.

[8] P. Chi *et al.*, "Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory," in *ISCA*, 2016.

[9] P. Chi, S. Li, C. Xu, T. Zhang, J. Zhao, Y. Liu, Y. Wang, and Y. Xie, "PRIME: a novel processing-in-memory architecture for neural network computation in ReRAM-based main memory," vol. 44, pp. 27–39.

[10] T. Chou, W. Tang, J. Botimer, and Z. Zhang, "CASCADE: Connecting RRAMs to extend analog dataflow in an end-to-end in-memory processing paradigm," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*. ACM.

[11] F. Devaux, "The true processing in memory accelerator," *2019 IEEE Hot Chips 31 Symposium (HCS)*, pp. 1–24, 2019.

[12] C. Eckert, X. Wang, J. Wang, A. Subramaniyan, R. Iyer, D. Sylvester, D. Blaaauw, and R. Das, "Neural cache: Bit-serial in-cache acceleration of deep neural networks," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, pp. 383–396.

[13] J. D. F. Ferreira, others, and O. Mutlu, "pluto: Enabling massively parallel computation in dram via lookup tables," in *ISCA*, 2021.

[14] D. Fujiki, S. Mahlke, and R. Das, "Duality cache for data parallel acceleration," in *Proceedings of the 46th International Symposium on Computer Architecture*, ser. ISCA '19. Association for Computing Machinery, pp. 397–410.

[15] J. Gómez-Luna, I. El Hajj, I. Fernandez, C. Giannoula, G. F. Oliveira, and O. Mutlu, "Benchmarking memory-centric computing systems: Analysis of real processing-in-memory hardware," in *2021 12th International Green and Sustainable Computing Conference (IGSC)*. IEEE, pp. 1–7.

[16] N. Hajinazar, G. F. Oliveira, S. Gregorio, J. D. Ferreira, N. M. Ghiasi, M. Patel, M. Alser, S. Ghose, J. Gómez-Luna, and O. Mutlu, "SIMDRAM: a framework for bit-serial SIMD processing using DRAM," in *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM.

[17] M. Imani, S. Gupta, Y. Kim, and T. Rosing, "FloatPIM: in-memory acceleration of deep neural network training with high precision," in *Proceedings of the 46th International Symposium on Computer Architecture*. ACM.

[18] S. Jeloka, N. B. Akesh, D. Sylvester, and D. Blaauw, "A 28 nm configurable memory (TCAM/BCAM/SRAM) using push-rule 6T bit cell enabling logic-in-memory," vol. 51, pp. 1009–1021.

[19] L. Ke, U. Gupta, B. Y. Cho, D. Brooks, V. Chandra, U. Diril, A. Firoozshahian, K. Hazelwood, B. Jia, H.-H. S. Lee, M. Li, B. Maher, D. Mudigere, M. Naumov, M. Schatz, M. Smelyanskiy, X. Wang, B. Reagen, C.-J. Wu, M. Hempstead, and X. Zhang, "RecNMP: Accelerating personalized recommendation with near-memory processing," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, pp. 790–803.

[20] H. Kwon, A. Samajdar, and T. Krishna, "MAERI: Enabling flexible dataflow mapping over DNN accelerators via reconfigurable interconnects," vol. 53, pp. 461–475.

[21] K. Lee, J. Jeong, S. Cheon, W. Choi, and J. Park, "Bit parallel 6T SRAM in-memory computing with reconfigurable bit-precision," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, pp. 1–6.

[22] S. Lee, S.-H. Kang, J. Lee, H. Kim, E. Lee, S. Seo, H. Yoon, S. Lee, K. Lim, H. Shin, J. Kim, O. Seongil, A. Iyer, D. Wang, K. Sohn, and N. S. Kim, "Hardware architecture and software stack for PIM based on commercial DRAM technology : Industrial product," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, pp. 43–56.

[23] S. Li *et al.*, "Pinatubo: A processing-in-memory architecture for bulk bitwise operations in emerging non-volatile memories," in *DAC*, 2016.

[24] S. Li, D. Niu, K. T. Malladi, H. Zheng, B. Brennan, and Y. Xie, "DRISA: a DRAM-based reconfigurable in-situ accelerator," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*. ACM.

[25] K. Liu *et al.*, "Recnmp: Accelerating personalized recommendation with near-memory processing," *arXiv*, 2021.

[26] O. Mutlu, S. Ghose, J. Gómez-Luna, and R. Ausavarungnirun, "Processing data where it makes sense: Enabling in-memory computation," vol. 67, pp. 28–41.

[27] G. F. Oliveira *et al.*, "Nim: An hmc-based machine for neuron computation," in *ARC*, 2017.

[28] G. F. Oliveira, M. Kabra, Y. Guo, K. Chen, A. G. Yağlıkçı, M. Soysal, M. Sadrosadati, J. O. Bueno, S. Ghose, J. Gómez-Luna, and O. Mutlu, "Proteus: Enabling high-performance processing-using-DRAM with dynamic bit-precision, adaptive data representation, and flexible arithmetic."

[29] G. F. Oliveira, A. Olgun, A. G. Yağlıkçı, F. N. Bostancı, J. Gómez-Luna, S. Ghose, and O. Mutlu, "MIMDRAM: An end-to-end processing-using-DRAM system for high-throughput, energy-efficient and programmer-transparent multiple-instruction multiple-data computing," in *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, pp. 186–203.

[30] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "XNOR-net: ImageNet classification using binary convolutional neural networks."

[31] V. Seshadri, D. Lee, T. Mullins, H. Hassan, A. Boroumand, J. Kim, M. A. Kozuch, O. Mutlu, P. B. Gibbons, and T. C. Mowry, "Ambit: in-memory accelerator for bulk bitwise operations using commodity DRAM technology," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*. ACM.

[32] H. Sharma, J. Park, N. Suda, L. Lai, B. Chau, J. K. Kim, V. Chandra, and H. Esmaeilzadeh, "Bit fusion: Bit-level dynamically composable architecture for accelerating deep neural network," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, pp. 764–775.

[33] F. A. Siddique, D. Guo, Z. Fan, M. Gholamrezaei, M. Baradaran, A. Ahmed, H. Abbot, K. Durrer, K. Nandagopal, E. Ermovick, K. Kiyawat, B. Gul, A. Mughrabi, A. Venkat, and K. Skadron, "Architectural modeling and benchmarking for digital DRAM PIM," in *2024 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, pp. 247–261.

[34] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition." [Online]. Available: http://arxiv.org/abs/1409.1556

[35] G. Singh, L. Chelini, S. Corda, A. Javed Awan, S. Stuijk, R. Jordans, H. Corporaal, and A.-J. Boonstra, "A review of near-memory computing architectures: Opportunities and challenges," in *2018 21st Euromicro Conference on Digital System Design (DSD)*. IEEE, pp. 608–617.

[36] A. Subramaniyan, J. Wang, E. R. M. Balasubramanian, D. Blaauw, D. Sylvester, and R. Das, "Cache automaton," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-50 '17. Association for Computing Machinery, pp. 259–272.

[37] J. Wang, X. Wang, C. Eckert, A. Subramaniyan, R. Das, D. Blaauw, and D. Sylvester, "A 28-nm compute SRAM with bit-serial logic/arithmetic operations for programmable in-memory vector computing," vol. 55, pp. 76–86. [Online]. Available: http://dx.doi.org/10.1109/JSSC.2019.2939682

[38] Z. Wang, C. Liu, A. Arora, L. John, and T. Nowatzki, "Infinity stream: Portable and programmer-friendly in-/near-memory fusion," in *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, ser. ASPLOS 2023. Association for Computing Machinery, pp. 359–375.

[39] I. E. Yuksel, others, and O. Mutlu, "Pulsar: Simultaneous many-row activation for reliable and high-performance computing in off-the-shelf dram chips," *arXiv*, 2023.

[40] J. Zhang, M. Imani, and E. Sadredini, "BP-NTT: Fast and compact in-SRAM number theoretic transform with bit-parallel modular multiplication."

[41] J. Zhang, H. Naghibijouybari, and E. Sadredini, "Sealer: In-SRAM AES for high-performance and low-overhead memory encryption," in *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design*. ACM, pp. 1–6.

[42] J. Zhang and E. Sadredini, "Inhale: Enabling high-performance and energy-efficient in-SRAM cryptographic hash for IoT," pp. 1–9.