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

Jingyao Zhang
University of California, Riverside
jzhan502@ucr.edu

Elaheh Sadredini
University of California, Riverside
elaheh@cs.ucr.edu

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. These layouts represent different execution models (word-parallel versus bit-serial) that directly determine how workloads utilize parallel compute columns and achieve load balance. 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 across both performance and energy dimensions. We develop iso-area, cycle-accurate BP and BS PIM architectural models with detailed energy accounting 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 in either performance or energy; the optimal choice is strongly dependent on workload characteristics. BP excels on control-flow-intensive tasks, achieving up to 14× latency and 29% energy advantages for arithmetic operations. Conversely, BS demonstrates energy-latency decoupling for logic-intensive kernels, delivering 6–59% energy savings even when cycle counts equal or exceed BP. 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 that optimize both performance and energy.

I. 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 [1]–[3]. 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 [5], and Tesseract [6]) and commercial systems (Samsung's HBM-PIM [7] and UPMEM [8]). 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 [9], DRISA [10], pLUTo [11], Fulcrum [12], FlexAmata [13], eAP [14], Impala [15], and PULSAR [16] demonstrate how bulk bitwise operations and table lookups can be efficiently executed in commodity DRAM [12]. Similarly, SRAM-based designs like Neural Cache [17], Compute Caches [18], and ASPEN [19] enable reconfigurable or bit-serial logic execution directly in SRAM arrays [20]–[23]. Non-volatile memory (NVM) technologies, such as ReRAM and PCM, also support compute-in-memory through techniques like PRIME [24] and Pinatubo [25], 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. Fundamentally, this is a parallel execution mapping problem where the layout choice directly affects how workloads utilize column-parallel resources, achieve load balance across columns, and incur predication overhead under control flow. The choice of data layout in PUM architectures has profound implications for performance and resource utilization across different workloads.

Many works adopt layout strategies driven by specific hardware constraints or workload assumptions, without systematic justification. For example, BitFusion [26] adopts a BS layout to exploit fine-grained bit-level reuse for DNN inference, while PRIME [27] uses a BP layout to enable analog matrix-vector multiplication in ReRAM crossbars. Ambit [9] supports both layouts depending on the type of operation but offers no frame-

work for guiding layout selection. Other architectures like BP-NTT [28], DRISA [10], Cascade [29], and FloatPIM [30] 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 [31], [32] 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 wastes cycles and increases energy per operation, highlighting 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 [34], 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 [35], [36] miss BP's advantages in control-flow-intensive code, while those focusing on traditional CPU benchmarks [37] 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. Several key questions remain unanswered, including whether edge-AI PIM should be bit-serial and when the overhead of supporting both layouts as in RecNMP [38] is worthwhile, which leads to ad-hoc and suboptimal design choices.

This paper presents the first systematic, workload-driven characterization of bit-parallel versus bit-serial data layouts in PIM architectures across both performance and energy dimensions. Rather than advocating for one layout over another, we demonstrate that **optimal layout selection is fundamentally workload-dependent in both latency and energy**, and that ignoring this dependency incurs substantial performance and energy penalties. Our key insight is that different computational patterns exhibit natural affinities to different layouts, and these affinities manifest differently across performance and energy metrics. 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 with detailed energy accounting for both BP and BS architectures under equal-area constraints, ensuring a fair comparison within identical silicon budgets. Unlike prior work that models only computation [39], we explicitly account for bit-transposition overhead in BS designs and provide comprehensive energy breakdowns across load, compute, and readout phases. We evaluate using a two-tier benchmark suite: (1) MIMDRAM microbenchmarks [40] that isolate specific computational patterns, and (2) PIMBench applications [33] 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 and Energy Analysis**: Through cycle-accurate modeling with detailed energy accounting of iso-area BP and BS designs, we quantify performance and energy differences across diverse workloads. We reveal up to 14× latency and 29% energy advantages for BP in arithmetic operations, while demonstrating that BS achieves 6–59% energy savings for logic-intensive kernels even when cycle counts equal or exceed BP. Furthermore, hybrid execution that dynamically switches between BP and BS achieves up to 2.66× speedup over the best static choice while optimizing energy efficiency.
- **Workload-Aware Design Framework**: We develop a comprehensive taxonomy that maps workload characteristics (parallelism degree, data precision, control com-

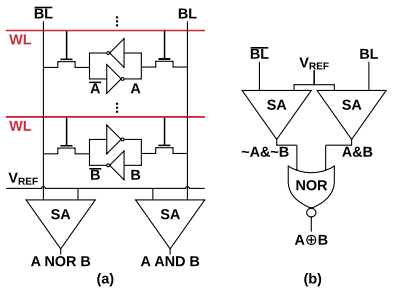


Fig. 1: In-SRAM bitline operations. Simultaneous activation of two wordlines accomplishes AND/NOR operations, while an extra NOR gate facilitates XOR operation.

plexity) and optimization objectives (latency vs. energy) 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 based on performance and energy requirements.

The remainder of this paper is organized as follows. Section II provides background on PIM architectures and data layouts. Section III presents our architectural analysis of BP versus BS designs. Section IV describes our experimental methodology. Section V presents our evaluation results across the benchmark suite. Section VI concludes.

II. 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 [7] and UPMEM [41]. 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], [27]. 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.

A. In-SRAM Computing Primitives

PUM architectures have been built on various memory technologies, including DRAM [9] and NVM [27]. 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 [42]. 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 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 [18] modifies the SA to support a richer set of logic operations, while Cache Automaton [43] uses a sense-amplifier cycling mechanism to accelerate specific computations.

Scope beyond SRAM: Although our analysis and models instantiate SRAM arrays for digital bitline-based computation, the core layout trade-offs (parallelism versus density, vertical storage pressure, lockstep control) follow from data layout geometry and execution model, rather than SRAM-specific circuit properties. We use SRAM as the baseline because it is a relatively neutral substrate with an approximately 1:1 read/write latency ratio, which minimizes technology-induced bias and helps isolate the layout-driven implications. We then compare multiple memory technologies in Section V-E.

B. 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 [18], Ambit [9], Sealer [44], BP-NTT [28], Sunder [45], and Inhale [46]. The BP approach is effective for both traditional SIMD-style inter-vector operations (EP-BP, e.g., adding two 16-element vectors of 32-bit integers in parallel) and for creating efficient intra-vector scratchpads (ES-BP, e.g., storing a sliding window of filter coefficients and state variables for FIR filters, where multiple intermediate results must remain resident).

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 [17], Duality Cache [47], SIMDRAM [48], MIMDRAM [40], and Infinity Stream [49].

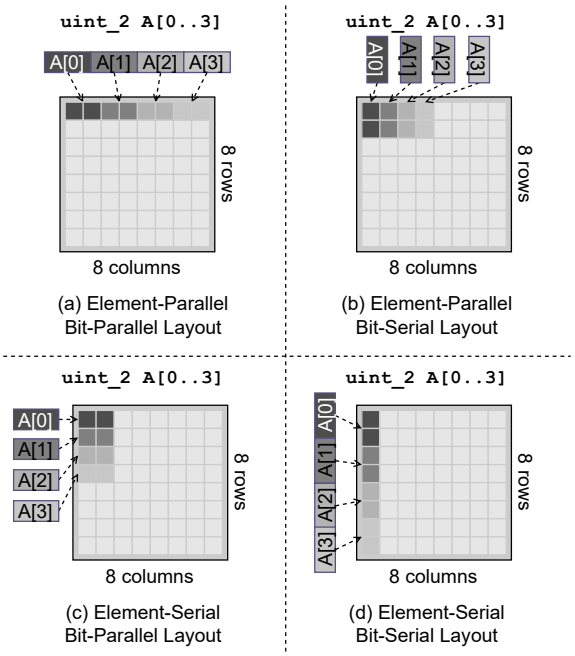


Fig. 2: The four hierarchical data layout schemes resulting from combining bit-level (BP, BS) and vector-level (EP, ES) organization. (a) EP-BP: Ideal for inter-vector operations on wide data. (b) EP-BS: Maximizes parallelism for inter-vector operations. (c) ES-BP: Efficiently buffers a single vector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

While highly effective for massively parallel inter-vector operations (EP-BS, e.g., computing XOR across 512 binary vectors simultaneously, with each column processing one vector independently), 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*.

Despite the prevalence of the BS paradigm, recent works like Proteus [50] 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.

III. Architectural Principles: Challenges

The Bit-Serial (BS) paradigm is foundational to many influential PUM systems, including Neural Cache [17], Duality Cache [47], SIMDRAM [48], MIMDRAM [40], and Infinity Stream [49], 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.

A. 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 [9], [18], are a physical dimension of 128 rows (`array_rows`) and 512 columns (`array_columns`) per array. **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 I 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 [51]. BS operates bit-by-bit, leveraging a 1-cycle hardware full adder for arithmetic and achieving zero-cost shifts by simply accessing adjacent rows [17]. 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.

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

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

B. 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.

Analysis. A 16-element vector-add has a DoP of 16. As illustrated conceptually in Figure 3b, a BS architecture must

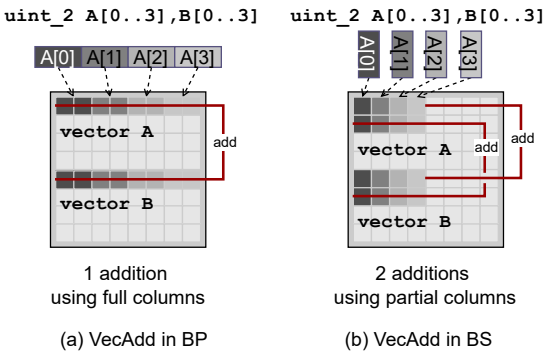


Fig. 3: Contrasting vector addition in (a) Bit-Parallel (BP) and (b) Bit-Serial (BS) layouts. A 4-element example is shown for clarity; the analysis generalizes to 16 elements. BP configures the array into wide PEs, utilizing the full array width. BS uses narrow 1-bit columns, leaving most of the hardware idle.

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. As a result, filtering efficiency depends on using the PIM array as a high-bandwidth, software-managed scratchpad.

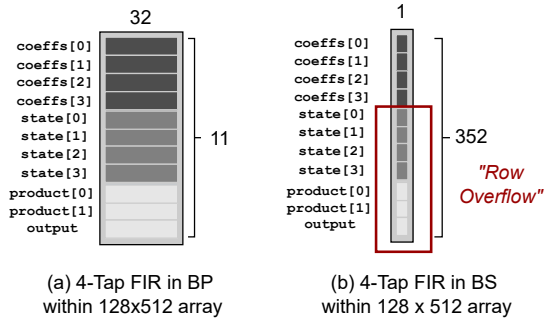


Fig. 4: Physical data layout for a 4-tap FIR filter. (a) In the BP layout, all required variables (coefficients, state, intermediate products, and the final output) are stored in separate rows, fitting within the array. (b) The BS layout attempts to store all these variables vertically, causing a massive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

Analysis. A 4-tap FIR filter requires the 4 'state' samples and 4 'coeffs' to be resident, plus 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.

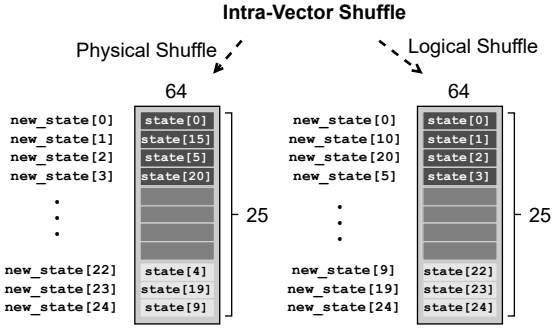
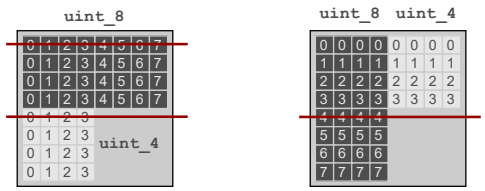


Fig. 5: Contrasting two permutation mechanisms. Left: a physical shuffle requires explicit data movement between memory locations, incurring multiple read-write cycles. Right: a logical shuffle achieves the same permutation through zero-cost address remapping, native to BP layouts with Element-Serial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

Fig. 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. Consider a simple conditional: `if (A < B) { result = D + E; } else { result = F * G; }`. Predicated execution transforms this into: (1) compute `sub = A - B`, (2) compute `mask` from sign bit, (3) compute `res_true = D + E`, (4) compute `res_false = F * G`, (5) select result using mask. This requires storing all input operands (A, B, D, E, F, G), intermediate results (`sub`, `mask`, `res_true`, `res_false`), and the final result, totaling at least 10 word-level variables. This transforms a control hazard into a data storage requirement, a trade-off that has profound implications for the underlying data layout.

Analysis. Predicated execution 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 II: Compute Cycle Latency for 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 II, 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.

C. 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: **(I) 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. **(II) 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. **(III) 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. **(IV) 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. Note that some BS limitations (e.g., limited working scratchpad) can be mitigated by increasing physical rows, but this increases array area, wire delay, and sensing energy, and does not address the root causes—vertical storage bottleneck, lockstep control conflict, and inherent bit-serial latency—so the qualitative trade-offs remain.

D. Where Bit-Serial Excels

The preceding analysis identifies scenarios where BS encounters fundamental limitations. However, BS exhibits strengths when workloads operate at low bitwidths or natively at the bit level, particularly when coupled with high DoP.

Case Study: PopCount & Binary Neural Networks. Two representative examples illustrate these advantages. First, population count, fundamental in bioinformatics, cryptography (Hamming weight), and sparse data structures, requires counting set bits in binary vectors. In the BS layout, each input vector is stored vertically in a single column, and popcount is performed through bit-by-bit logic operations followed by accumulation. When processing multiple vectors, each occupies a separate column, enabling all available columns to operate in parallel. In contrast, BP must partition each vector into multiple words and perform popcount within each word separately, which reduces bit-level parallelism and limits the number of vectors that can be processed simultaneously. Second, Binary Neural Networks (BNNs) quantize both weights and activations to 1-bit values [34], with XNOR-followed-by-popcount as the core operation. For 1-bit data, the previously discussed challenges largely disappear: vertical storage overhead becomes negligible (each element occupies a single cell), underutilization vanishes when the workload provides sufficient vectors to fill all columns, and high per-operation latency is amortized across massive column-level parallelism. These examples illustrate that optimal layout selection depends on workload characteristics such as DoP, data bitwidth, and whether operations are bit-level or word-level.

E. Implications for Hybrid Approaches

The complementary strengths of BP and BS motivate hybrid architectures that support both layouts and enable dynamic selection based on workload characteristics. However, switching between layouts incurs transposition overhead: converting data between BP's horizontal word organization and BS's vertical bit organization requires explicit read-transpose-write operations. For example, transposing a 32×32 bit matrix requires reading 32 rows, performing a bitwise transpose, and writing back 32 columns, incurring significant latency and energy costs. This overhead must be amortized over sufficiently long compute phases to justify layout switching. We evaluate hybrid execution strategies and quantify when the benefits of layout-specific optimizations outweigh transposition costs in Section V.

IV. 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 III to precisely quantify performance across a diverse set of computational patterns, foregoing traditional hardware simulation for a more direct, model-based analysis.

A. Architectural Model

Our study targets a single 128 row $\times$ 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.* [51], is shared; only the peripheral logic differs, ensuring an *iso–area* comparison.

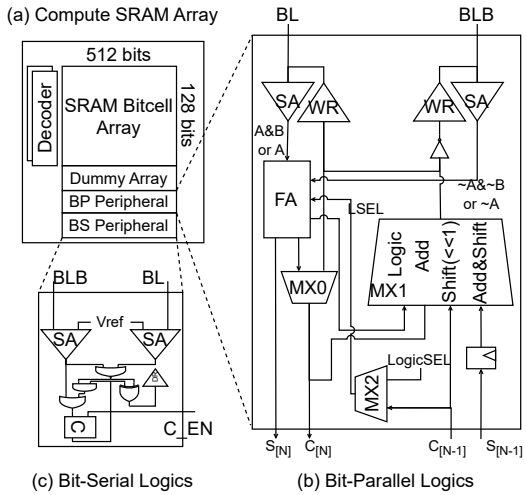


Fig. 7: Compute SRAM array *(a)* with dual peripherals: a word–level BP datapath *(b)* and a 1–bit BS datapath *(c)* sharing the same 128 $\times$ 512 cell core. Mode selection is performed by a lightweight mux on each column sense line.

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 I. 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 [17]. 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.

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 [17], [18], [51]. 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 [48]. 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 read$(M) + 1 +$ write(N) cycles in the BP$\rightarrow$BS direction and read$(N) + 1 +$ write(M) in the reverse direction. These costs match the per–round accounting used in our AES study (Section V-D).

B. Model Validation

Our primitive cycle costs are grounded in measured or reported silicon data from SRAM compute peripheries and in–array logic [17], [48], [51]. We validate our analytical totals against two public benchmark suites, MIMDRAM [40] (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.

C. 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.

1) Tier 1: Microbenchmarks: To isolate the performance of fundamental computational patterns, we select a broad set of microbenchmarks inspired by the MIMDRAM [40] 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.

2) Tier 2: Application Benchmarks: To assess performance on real-world workloads [52]–[57], 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.

V. Evaluation

A. Evaluation Goals

Section III 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 three questions: **Q1:** How do BP and BS compare across arithmetic, logic, reduction, and control kernels in latency and energy, and which root causes dominate? **Q2:** How do storage density and batching affect end-to-end time as datasets exceed one array? **Q3:** At the application level, when do static BP/BS or hybrid switching (with transpose overheads) win?

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.

Evaluation Strategy. Our evaluation systematically targets each challenge from Section III. Table IV organizes micro-kernels by challenge: arithmetic operations test Challenge 6 (high word-level latency); bit-manipulation kernels address Challenge 1 (underutilization under low DoP); control/predicate kernels examine Challenges 2 and 4 (row overflow and mixed-precision conflicts). Challenge 3 (inefficient shuffles) and Challenge 5 (control flow overhead) are evaluated through application benchmarks such as AES and Radix-Sort, where phase diversity necessitates hybrid execution. Application benchmarks validate these findings at scale and quantify hybrid layout trade-offs with transposition overhead.

B. Experimental Setup

Hardware Assumptions. Iso-area BP and BS arrays (Section IV); baseline 128 rows, 512 columns; costs from Table I. Differences arise only from data layout and execution style. **Benchmark Suite.** Two tiers (Section IV-C): 18 micro-kernels (two sizes: 1 024, 65 536) and 22 applications (vision, learning, analytics, crypto) with standard datasets and identical inputs across layouts. **Methodology.** Total cycles = load + compute + readout; compute from Table I. Energy from BP-IMC [51] and BS-IMC [58], [59]. Load/readout scale with active

rows (captures batching). Runtimes normalized to 1 GHz and reported in cycles.

C. Tier–1: Microbenchmark Analysis

Our microbenchmark analysis proceeds in two steps, using precise data from Tables IV and III. 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.

a) Performance at Fixed Scale: Table IV shows the core trends succinctly. Arithmetic kernels (Challenge 6) favor BP's word-level datapath in both latency and energy (e.g., `Vector Add` in 1–2 cycles; `MULTU` 18 vs. 256 cycles). Bit-centric kernels (Challenge 1) (e.g., `bitcount`, native `Reduction`) tend to favor BS due to fine-grained parallelism. Predicate-heavy code (Challenges 2, 4) exposes energy–latency decoupling: BS may use more cycles yet less energy by avoiding wide masks; BS also risks row overflow in predicated forms, whereas BP typically fits within available rows.

b) Energy Efficiency Analysis: Energy and latency need not align. Logic/predicate kernels often let BS save energy despite equal or higher cycle counts (e.g., `ReLU`, `ge_0`/`gt_0`); arithmetic kernels usually favor BP in both metrics (e.g., `MULTU`). The rule of thumb: many cheap 1-bit operations can beat a few expensive word operations.

c) Sensitivity to Workload Size: End-to-end performance is sensitive to workload size due to BP's lower storage density. Table III uses vector addition to illustrate batching.

TABLE III: 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×

At 1K elements, BP is 1.15× faster; at 16K (single-batch limit) the benefit shrinks to 1.01× as load/readout grow. At 64K, BP needs 4 batches and loses its compute advantage, matching BS. This reflects the trade-off between BP's low compute latency (Challenge 6) and lower storage density (Challenge 2).

d) Synthesis: Optimal layout depends on kernel traits, size, and objective (latency vs. energy). Small-scale arithmetic: BP wins (up to 14× faster, 29% less energy); bit-level tasks: BS can win (e.g., `bitcount`). Energy–latency decoupling is common for logic/predicate kernels. As size grows, BP's batching erodes its compute edge, motivating workload-aware layout choices.

e) Implication—batching neutralizes advantages: BP's latency advantage holds only within a single batch; once batching kicks in, load/readout dominate and BP converges toward BS—explaining smaller application-level speedups and motivating workload-aware choices.

D. Tier–2: Application Benchmark Results

We profile 22 applications (vision, learning, analytics, crypto) and report total kernel cycles under BP/BS as described in Section V-B. To expose ample parallelism, we assume **512 parallel arrays**. Table V groups workloads by speedup and limiting factor.

a) Case study 1: VGG-13 inference (utilization): VGG-13 [60] shows utilization varying with shrinking feature maps (Figure 8, capacity 262,144 bits, 16-bit elements, 3×3 reuse).

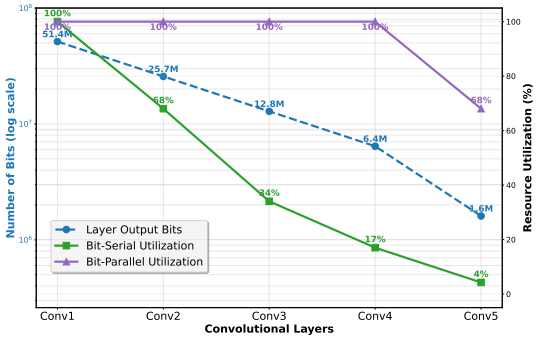


Fig. 8: VGG-13 layer output size and PIM resource utilization. Left axis (log scale) shows layer output bits versus PIM capacity. Right axis shows the resulting resource utilization for BP and BS.

Early layers (Conv1–3) saturate capacity for both; differences come from BP's lower compute latency. Later layers (Conv4–5) underutilize BS (17%, 4%) due to limited parallelism, while BP remains high (100%, 68%) via word-level granularity.

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.

b) 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 VI 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 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 [61] that transforms the S-box into just 115

TABLE IV: Cycle and energy breakdown of advanced micro-kernels. Rows/Elem and Cols/Elem denote physical footprint per element in the array. All values are for 16-bit data.

Kernel	Variant	Mode	Rows/Elem	Cols/Elem	Load	Compute	Readout	Total	Load (nJ)	Compute (nJ)	Readout (nJ)	Total (nJ)
Arithmetic kernels (Challenge 6: inherent computational latency)												
Vector Add	Standard	BP	~3	16	64	1	32	**97**	0.721	0.563	0.360	**1.64**
		BS	49	1	64	16	32	112	1.15	0.898	0.577	2.63
Vector Sub	Standard	BP	~3	16	64	2	32	**98**	0.721	1.12	0.360	**2.20**
		BS	49	1	64	16	32	112	1.15	0.898	0.577	2.63
MULTU	Parallel	BP	~4	32	128	18	64	**210**	0.721	10.2	0.721	**11.7**
	Shift+Add	BS	64	1	64	256	64	384	1.15	14.4	1.15	16.7
MULTU Const	Parallel	BP	~3	32	64	18	64	**146**	0.361	10.2	0.721	**11.3**
	Shift+Add	BS	48	1	32	256	64	352	0.577	14.4	1.15	16.1
DIVU	Restoring	BP	4	16	64	640	32	**736**	0.721	39.8	0.360	**40.9**
	Restoring	BS	64	1	64	1280	32	1376	1.15	62.9	0.577	64.6
MIN	Shift Mask	BP	~5	16	64	21	32	**117**	0.721	2.34	0.360	**3.42**
	Iter. Comp.	BS	50	1	64	96	32	192	1.15	4.33	0.577	6.06
MAX	Shift Mask	BP	~5	16	64	21	32	**117**	0.721	2.34	0.360	**3.42**
	Iter. Comp.	BS	50	1	64	96	32	192	1.15	4.33	0.577	6.06
Logical / Bit–manipulation kernels (Challenge 1: granularity mismatch)												
Reduction	Tree	BP	2	16	32	19	16	67	0.360	0.422	2.2e-2	**0.805**
	Native	BS	17	1	32	16	16	**64**	0.577	0.721	3.6e-2	1.33
bitcount	D&C	BP	3	16	128	25	32	185	1.44	6.19	0.360	7.99
	Summation	BS	26	1	32	80	16	**128**	0.577	4.49	0.288	**5.35**
bitweave	1b Logic	BP	53	1024	96	225	2	**323**	2.3e-2	1.1e-3	1.4e-3	**2.5e-2**
	2b Logic	BS	74	512	64	434	2	500	7.3e-2	3.5e-3	3.6e-2	0.113
	4b Logic	BS	116	256	48	852	2	902	0.145	7.0e-3	3.6e-2	0.188
Control / Predicate kernels (Challenge 4: lock-step control; Challenge 2: vertical storage bottleneck in some cases)												
abs	Shift Mask	BP	3	16	32	18	32	**82**	0.360	2.34	0.360	**3.06**
	Serialised	BS	48	1	32	48	32	112	0.577	2.34	0.577	3.49
if–then–else	Mask 0-s	BP	5	16	96	7	32	**135**	1.08	3.37	0.360	4.81
	Synth. MUX	BS	52	1	80	49	32	161	1.44	2.21	0.577	**4.23**
equal	XOR+Reduce	BP	3	16	64	22	32	**118**	0.721	1.35	0.360	**2.43**
	Serial XOR	BS	49	1	64	33	32	129	1.15	1.49	0.577	3.22
ge_0	Shift	BP	1	16	32	17	16	65	0.360	1.67	2.2e-2	2.05
	Sign Bit	BS	16	1	32	1	16	**49**	0.577	4.5e-2	0.288	**0.91**
gt_0	Synth.	BP	3	16	32	35	32	99	0.360	4.00	0.360	4.72
	Serial Red.	BS	17	1	32	17	32	**81**	0.577	0.766	0.577	**1.92**
ReLU	Standard	BP	~3	16	32	17	32	81	0.360	1.32	0.360	2.04
	Standard	BS	49	1	32	17	32	81	0.577	0.766	0.577	**1.92**

TABLE V: 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 neutralizes 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 VI: Per-round cycle breakdown for AES-128.

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	-

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 (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 (Per round: 16 + 115 + 32 + 272 + 290 = 725 cycles), where the 290-cycle transposition overhead (two transposes per round) is amortized across the round.

Sensitivity to transpose cost. The hybrid win holds even with 10× slower transpose; total runtime rises only ∼2.6%, with 2.59× speedup vs. BP.

For AES-128 (10 rounds): BP = 18,624 cycles, BS = 26,750, Hybrid = 6,994 (**2.66**× over BP), as SubBytes strongly favors BS while other stages favor BP and transpose cost is modest.

c) Takeaway: Application results corroborate the kernel analysis: (1) BP's latency advantage translates to ∼ 1.3× median speedup for arithmetic-heavy workloads; (2) bit-centric tasks favor 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.

d) Energy efficiency considerations: Three patterns emerge: (1) Correlation—arithmetic favors BP in both latency and energy; (2) Decoupling—logic/predicate kernels can let BS save energy despite equal/greater cycles; (3) Inversion—sign-bit predicates often let BS win in both metrics.

These patterns have profound implications for energy-

TABLE VII: Cross-technology sensitivity analysis using microbenchmark cycles from Table IV. Speedup is reported as BS latency / BP latency (i.e., BP speedup over BS); higher values indicate BP performs better.*

Technology	Write/Read Ratio	VecAdd (BS/BP)	Bitcount (BS/BP)
SRAM [62]	1.00×	1.15×	0.69×
DRAM [63]	1.35×	1.17×	0.70×
ReRAM [27]	2.26×	1.19×	0.71×
LL-PCM [64]	4.67×	1.21×	0.72×
PCM [64]	5.40×	1.21×	0.72×

*Energy write/read ratios: SRAM (1.0×) [62], DRAM (1.2×) [63], ReRAM (8.5×) [65], LL-PCM (15.3×) [64], PCM (138.5×) [64]. For NVM technologies, energy asymmetry can exceed latency asymmetry, further weighting the write component in the energy dimension.

constrained PIM systems. In battery-powered edge devices where energy budgets are strict, the `if-then-else` case study suggests that accepting modest latency penalties (19%) for substantial energy gains (12%) may be preferable. Conversely, datacenter PIM accelerators optimizing for throughput should prioritize BP for arithmetic workloads despite slightly higher energy costs.

E. Cross-technology analysis

Although our core evaluation uses SRAM, the BP/BS trade-offs primarily follow data layout geometry and the execution model rather than SRAM-specific circuit details. To extend these results, we ask how a technology-dependent write/read asymmetry changes the relative cost of the phases in Table IV.

Let t_r and t_w denote read and write latencies, and $\rho \triangleq t_w/t_r$. We interpret the cycle counts in Table IV as event counts, and model the normalized end-to-end time for layout $L \in \{\mathrm{BP}, \mathrm{BS}\}$ as $T^{(L)}(\rho) = C_r^{(L)} + \rho\, C_w^{(L)}$, where C_r corresponds to Readout and C_w to Load+Compute (many PUM primitives are effectively read–modify–write [17], [18], [47]). We report speedup as $S(\rho) = T^{(\mathrm{BS})}(\rho)/T^{(\mathrm{BP})}(\rho)$.

Table VII shows representative trends. When one layout has lower C_w, increasing ρ amplifies that advantage; when one layout has both lower C_w and lower C_r, the relative margin changes modestly. In most cases, varying ρ does not reverse the preferred layout, as $S(\rho)$ approaches $C_w^{(\mathrm{BS})}/C_w^{(\mathrm{BP})}$ for large ρ. A crossover requires that one layout has lower C_w but higher C_r, creating a finite ρ^* where $T^{(\mathrm{BS})}(\rho^*) = T^{(\mathrm{BP})}(\rho^*)$. For single-kernel microbenchmarks, this pattern is uncommon (the layout with fewer update steps typically also reads out fewer or comparable output bits), and the preferred layout remains consistent across the studied ρ range. In the energy dimension, where write/read asymmetry can be stronger, the same weighting effect applies.

F. Synthesis and Key Takeaways

The evaluation tiers paint a coherent picture: neither data layout dominates; instead, each excels under specific workload conditions. Table VIII summarises the qualitative trends distilled from more than forty quantitative data points. **Hybrid data layouts:** with fast on-chip transpose, dynamic switching can beat the best static layout whenever phases alternately favor BS and BP (profitable when transpose is $< 2\%$ of per-phase time). **Future work:** explore row-selective transpose units and compiler analyses that insert transposes only when predicted gains exceed hardware thresholds. In summary, our evaluation confirms the central thesis that *data layout is a first-order design decision for PIM in both performance and energy*. A workload-aware architecture that can flexibly navigate the BP/BS continuum, guided by latency requirements, energy budgets, and workload characteristics, can unlock significant and otherwise untapped efficiency in both dimensions. The methodology and empirical findings presented here provide a quantitative foundation for building such adaptive systems.

TABLE VIII: Workload characteristics favoring each layout.

BP-friendly	BS-friendly
Word-level arithmetic (add/-mul/div)	Bit-level operations (pop-count, 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

VI. 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 four key insights: **Architecture matters.** BP achieves up to 14× lower latency and 29% better energy efficiency than BS for arithmetic kernels by eliminating serial carry propagation and enabling word-level parallelism. **Energy-latency decoupling.** For logic-intensive operations, BS can deliver 6–59% energy savings even when cycle counts equal or exceed BP, through fine-grained 1-bit operations that avoid costly word-level computations. **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, optimizing both performance and energy. These findings transform data layout from an implicit assumption into a first-class design parameter for both performance and energy optimization. Future PIM systems should support low-cost transposition hardware and workload-aware compilers that orchestrate dynamic layout decisions based on latency requirements, energy budgets, and workload characteristics. By establishing that neither BP nor BS is universally optimal across both performance and energy dimensions, this work motivates adaptive architectures that navigate the BP/BS continuum to unlock efficiency gains in both metrics.

Acknowledgment

This work is funded, in part, by NSF Career Award #2339317, NSF #2235398, and the initial complement fund from UCR.

REFERENCES

[1] A. Boroumand *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*, 2018.

[2] S. G. Ghinani, J. Zhang, and E. Sadredini, "Enabling low-cost secure computing on untrusted in-memory architectures," in *Proceedings of the Security Symposium*, 2025.

[3] Z. Li *et al.*, "Efficient algorithms for task mapping on heterogeneous cpu/gpu platforms for fast completion time," *Journal of Systems Architecture*, vol. 114, p. 101936, 2021.

[4] H. Asghari-Moghaddam *et al.*, "Chameleon: Versatile and practical near-dram acceleration architecture for large memory systems," in *Proceedings of the International Symposium on Microarchitecture*, 2019.

[5] L. Ke *et al.*, "Recnmp: Accelerating personalized recommendation with near-memory processing," in *Proceedings of the International Symposium on Computer Architecture*, 2020, pp. 790–803.

[6] J. Ahn *et al.*, "A scalable processing-in-memory accelerator for parallel graph processing," in *Proceedings of the International Symposium on Computer Architecture*, 2015.

[7] S. Lee *et al.*, "Hardware architecture and software stack for PIM based on commercial DRAM technology : Industrial product," in *Proceedings of the International Symposium on Computer Architecture*, 2021, pp. 43–56.

[8] F. Devaux, "The true processing in memory accelerator," in *Proceedings of the Hot Chips Symposium*, 2019, pp. 1–24.

[9] V. Seshadri *et al.*, "Ambit: in-memory accelerator for bulk bitwise operations using commodity DRAM technology," in *Proceedings of the 50th Annual International Symposium on Microarchitecture*, 2017.

[10] S. Li *et al.*, "DRISA: a DRAM-based reconfigurable in-situ accelerator," in *Proceedings of the 50th Annual International Symposium on Microarchitecture*, 2017.

[11] J. D. Ferreira *et al.*, "pLUTo: Enabling massively parallel computation in DRAM via lookup tables," in *Proceedings of the International Symposium on Computer Architecture*, 2021.

[12] M. Lenjani *et al.*, "Fulcrum: A simplified control and access mechanism toward flexible and practical in-situ accelerators," in *Proceedings of the International Symposium on High Performance Computer Architecture*, 2020, pp. 556–569.

[13] E. Sadredini *et al.*, "FlexAmata: A universal and efficient adaption of applications to spatial automata processing accelerators," in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2020, pp. 219–234.

[14] E. Sadredini *et al.*, "eAP: A scalable and efficient in-memory accelerator for automata processing," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2019, pp. 87–99.

[15] E. Sadredini *et al.*, "Impala: Algorithm/architecture co-design for in-memory multi-stride pattern matching," in *2020 IEEE international symposium on high performance computer architecture (HPCA)*. IEEE, 2020, pp. 86–98.

[16] I. E. Yuksel *et al.*, "Simultaneous many-row activation in off-the-shelf dram chips: Experimental characterization and analysis," *arXiv preprint arXiv:2405.06081*, 2024.

[17] C. Eckert *et al.*, "Neural cache: Bit-serial in-cache acceleration of deep neural networks," in *Proceedings of the International Symposium on Computer Architecture*, 2018, pp. 383–396.

[18] S. Aga *et al.*, "Compute caches," in *Proceedings of the International Symposium on High Performance Computer Architecture*, 2017, pp. 481–492.

[19] K. Angstadt *et al.*, "ASPEN: A scalable in-sram architecture for pushdown automata," in *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2018, pp. 921–932.

[20] J. Zhang and E. Sadredini, "Unlocking energy-efficient and high-throughput secure data communication in IoT with memory-centric computing," in *2025 International Parallel and Distributed Processing Symposium Workshops*, 2025, pp. 1314–1315.

[21] J. Zhang and E. Sadredini, "CryptoSRAM: Enabling high-throughput cryptography on MCUs via in-SRAM computing," *arXiv preprint arXiv:2509.22986*, 2025.

[22] J. Zhang and E. Sadredini, "A near-cache architectural framework for cryptographic computing," *arXiv preprint arXiv:2509.23179*, 2025.

[23] J. Zhang *et al.*, "SAIL: SRAM-accelerated LLM inference system with lookup-table-based GEMV," *arXiv preprint arXiv:2509.25853*, 2025.

[24] P. Chi *et al.*, "Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory," in *Proceedings of the International Symposium on Computer Architecture*, 2016.

[25] S. Li *et al.*, "Pinatubo: A processing-in-memory architecture for bulk bitwise operations in emerging non-volatile memories," in *Proceedings of the Design Automation Conference*, 2016.

[26] H. Sharma *et al.*, "Bit fusion: Bit-level dynamically composable architecture for accelerating deep neural network," in *Proceedings of the International Symposium on Computer Architecture*, 2018, pp. 764–775.

[27] P. Chi *et al.*, "PRIME: a novel processing-in-memory architecture for neural network computation in ReRAM-based main memory," in *Proceedings of the International Symposium on Computer Architecture*, 2016, pp. 27–39.

[28] J. Zhang, M. Imani, and E. Sadredini, "BP-NTT: Fast and compact in-SRAM number theoretic transform with bit-parallel modular multiplication," in *Proceedings of the Design Automation Conference*, 2023, pp. 1–6.

[29] T. Chou *et al.*, "CASCADE: Connecting RRAMs to extend analog dataflow in an end-to-end in-memory processing paradigm," in *Proceedings of the International Symposium on Microarchitecture*, 2019, pp. 114–125.

[30] M. Imani *et al.*, "FloatPIM: in-memory acceleration of deep neural network training with high precision," in *Proceedings of the 46th International Symposium on Computer Architecture*, 2019, pp. 802–815.

[31] O. Mutlu *et al.*, "Processing data where it makes sense: Enabling in-memory computation," *Microprocessors and Microsystems*, vol. 67, pp. 28–41, 2019.

[32] G. Singh *et al.*, "A review of near-memory computing architectures: Opportunities and challenges," in *Proceedings of the Euromicro Conference on Digital System Design*, 2018, pp. 608–617.

[33] F. A. Siddique *et al.*, "Architectural modeling and benchmarking for digital DRAM PIM," in *2024 International Symposium on Workload Characterization*, 2024, pp. 247–261.

[34] M. Rastegari *et al.*, "XNOR-net: ImageNet classification using binary convolutional neural networks," *arXiv preprint arXiv:1603.05279*, 2016.

[35] Y.-H. Chen, J. Emer, and V. Sze, "Eyeriss: a spatial architecture for energy-efficient dataflow for convolutional neural networks," *SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 367–379, 2016.

[36] H. Kwon, A. Samajdar, and T. Krishna, "MAERI: Enabling flexible dataflow mapping over DNN accelerators via reconfigurable interconnects," *SIGPLAN Notices*, vol. 53, no. 2, pp. 461–475, 2018.

[37] J. Ahn *et al.*, "PIM-enabled instructions: a low-overhead, locality-aware processing-in-memory architecture," *SIGARCH Computer Architecture News*, vol. 43, no. 3S, pp. 336–348, 2016.

[38] L. Ke *et al.*, "RecNMP: Accelerating personalized recommendation with near-memory processing," in *Proceedings of the International Symposium on Computer Architecture*, 2020, pp. 790–803.

[39] 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*, 2018.

[40] G. F. Oliveira *et al.*, "MIMDRAM: An end-to-end processing-using-DRAM system for high-throughput, energy-efficient and programmer-transparent multiple-instruction multiple-data computing," in *Proceedings of the International Symposium on High-Performance Computer Architecture*, 2024, pp. 186–203.

[41] J. Gómez-Luna *et al.*, "Benchmarking memory-centric computing systems: Analysis of real processing-in-memory hardware," in *Proceedings of the International Green and Sustainable Computing Conference*, 2021, pp. 1–7.

[42] S. Jeloka *et al.*, "A 28 nm configurable memory (TCAM/BCAM/SRAM) using push-rule 6T bit cell enabling logic-in-memory," *Journal of Solid-State Circuits*, vol. 51, no. 4, pp. 1009–1021, 2016.

[43] A. Subramaniyan *et al.*, "Cache automaton," in *Proceedings of the 50th Annual International Symposium on Microarchitecture*, 2017, pp. 259–272.

[44] J. Zhang, H. Naghibijouybari, and E. Sadredini, "Sealer: In-SRAM AES for high-performance and low-overhead memory encryption," in *Proceedings of the International Symposium on Low Power Electronics and Design*, 2022, pp. 1–6.

[45] E. Sadredini *et al.*, "Sunder: Enabling low-overhead and scalable near-data pattern matching acceleration," in *MICRO-54: 54th Annual*

IEEE/ACM International Symposium on Microarchitecture, 2021, pp. 311–323.

[46] J. Zhang and E. Sadredini, "Inhale: Enabling high-performance and energy-efficient in-SRAM cryptographic hash for IoT," in *Proceedings of the 41st International Conference on Computer-Aided Design*, 2022, pp. 1–9.

[47] D. Fujiki, S. Mahlke, and R. Das, "Duality cache for data parallel acceleration," in *Proceedings of the 46th International Symposium on Computer Architecture*, 2019, pp. 397–410.

[48] N. Hajinazar *et al.*, "SIMDRAM: a framework for bit-serial SIMD processing using DRAM," in *Proceedings of the 26th International Conference on Architectural Support for Programming Languages and Operating Systems*, 2021.

[49] Z. Wang *et al.*, "Infinity stream: Portable and programmer-friendly in-/near-memory fusion," in *Proceedings of the 28th International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, 2023, pp. 359–375.

[50] G. F. de Oliveira Junior *et al.*, "Proteus: Achieving high-performance processing-using-dram with dynamic bit-precision, adaptive data representation, and flexible arithmetic," in *Proceedings of the 39th International Conference on Supercomputing*, 2025, p. 473–494.

[51] K. Lee *et al.*, "Bit parallel 6T SRAM in-memory computing with reconfigurable bit-precision," in *Proceedings of the Design Automation Conference*, 2020, pp. 1–6.

[52] S. Wu *et al.*, "Turbofno: High-performance fourier neural operator with fused fft-gemm-ifft on gpu," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2025, pp. 991–1005.

[53] S. Wu *et al.*, "Turbofft: Co-designed high-performance and fault-tolerant fast fourier transform on gpus," in *Proceedings of the 30th SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming*, 2025, pp. 70–84.

[54] S. Wu *et al.*, "Ft k-means: A high-performance k-means on gpu with fault tolerance," in *2024 International Conference on Cluster Computing*, 2024, pp. 322–334.

[55] S. Wu *et al.*, "Anatomy of high-performance gemm with online fault tolerance on gpus," in *Proceedings of the 37th International Conference on Supercomputing*, 2023, pp. 360–372.

[56] J. Liu *et al.*, "Cusz-i: High-ratio scientific lossy compression on gpus with optimized multi-level interpolation," in *SC24: International Conference for High Performance Computing, Networking, Storage and Analysis*, 2024, pp. 1–15.

[57] S. Wu *et al.*, "Boosting scientific error-bounded lossy compression through optimized synergistic lossy-lossless orchestration," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2025, pp. 2024–2037.

[58] J. Wang *et al.*, "A 28-nm compute SRAM with bit-serial logic/arithmetic operations for programmable in-memory vector computing," *Journal of Solid-State Circuits*, vol. 55, no. 1, pp. 76–86, 2020.

[59] K. Al-Hawaj *et al.*, "EVE: Ephemeral vector engines," in *Proceedings of the International Symposium on High-Performance Computer Architecture*, 2023, pp. 691–704.

[60] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *3rd International Conference on Learning Representations*, 2015, pp. 1–14.

[61] J. Boyar and R. Peralta, "A new combinational logic minimization technique with applications to cryptology," in *Experimental Algorithms*. Springer Berlin Heidelberg, 2010, pp. 178–189.

[62] Alliance Memory Inc., "5V 1Mb (128K x 8) fast asynchronous sram." [Online]. Available: https://www.mouser.com/datasheet/2/12/AS7C1024B_March2022_v3_0-3680758.pdf

[63] "K4A8G165WC 8Gb C-die DDR4 SDRAM x16 datasheet." [Online]. Available: https://semiconductor.samsung.com/resources/user-manual/x16%20only_8G_C_DDR4_Samsung_Spec_Rev1.5_Apr.17.pdf

[64] N. S. Kim *et al.*, "LL-PCM: Low-latency phase change memory architecture," in *Proceedings of the Design Automation Conference*, 2019.

[65] M. Mao *et al.*, "Programming strategies to improve energy efficiency and reliability of ReRAM memory systems," in *2015 Workshop on Signal Processing Systems*, 2015, pp. 1–6.

APPENDIX: ARTIFACT DESCRIPTION

This appendix follows the IPDPS Artifact Description (AD) structure and documents the open-source components released for this paper.

Overview of Contributions and Artifacts

Paper's main contributions (summary).

- C_1 A systematic, workload-driven characterization of bit-parallel (BP) vs. bit-serial (BS) PIM data layouts in latency and energy.
- C_2 A tiered evaluation methodology that links kernel-level behaviors to application-level implications.
- C_3 Actionable guidelines for selecting BP/BS layouts (and motivating hybrid switching).

Computational artifacts.

- A_1 **Open-source microkernel model (cycles & energy).** GitHub: github.com/jingyao-zhang/NOSFA. DOI: 10.5281/zenodo.18364410. License: MIT.

Artifact–contribution mapping.

Artifact ID	Contributions supported	Related paper elements
A_1	C_1, C_2	Table IV; Table III.

Artifact Identification: A_1 (Microkernel model)

Relationship to contributions. A_1 provides the reference implementation used to compute the microkernel cycle and energy (fJ) breakdowns for BP/BS layouts, supporting the kernel-level evidence behind our characterization and methodology.

Expected outcomes. Running A_1 regenerates the microkernel breakdown table values (cycles and fJ) and verifies them against the reference table shipped with the artifact.

Time to reproduce.

- **Setup:** ∼1–2 minutes (clone repository; ensure Python 3).
- **Execution:** ∼10–30 seconds (run all analyzers).
- **Analysis:** ∼1 minute (inspect generated table and comparison report).

Requirements.

- **Hardware:** any modern x86/ARM CPU machine (laptop/desktop).
- **Software:** Python 3 (standard library only); no third-party packages required.
- **Datasets / inputs:** none.
- **Installation / deployment:** no compilation; run scripts directly.

Experiment workflow.

- T_1 **Generate results:** execute all `microkernels/*_analyzer.py` and collect rows into a single table.
- T_2 **Validate results:** compare regenerated values against `microkernels/results.md`.

Dependencies: $T_1 \rightarrow T_2$.

How to run. From `microkernels_open_source/`:

```
python scripts/generate_results_md.py -out
generated_results.md
python scripts/compare_results.py
-ref microkernels/results.md -gen
generated_results.md
```

Outputs. The artifact produces `generated_results.md` and prints a comparison summary indicating whether regenerated rows match the reference values (up to benign formatting differences such as commas or variant labels).

Scope and limitations (disclosure). *This open-source release focuses on Tier–1 microkernels (cycle/energy modeling). Some Tier–2 application-level results in Section V-D are derived via analytical composition and manual post-processing described in the paper, and we do not provide a fully automated end-to-end pipeline for regenerating all application figures/tables in this artifact.*

Planned updates. We plan to extend the public package with additional automation as it becomes available.