Modeling Tradeoffs of Error Correction Schemes in Processing-in-Memory Architectures

Michael Shires
University of Virginia
jrs6qe@virginia.edu

Kelly Farran
University of Virginia
wds8wd@virginia.edu

Derek Hansen
University of Virginia
hkk6pa@virginia.edu

***Abstract*—Processing-in-Memory (PIM) can improve performance by reducing data movement between DRAM and the host processor, but server-grade deployments demand reliability guarantees that existing PIM simulation frameworks do not model. We extend the simulation framework PIMeval-PIMbench with a three-tier ECC hierarchy:JEDEC-mandated on-die DRAM ECC, device-to-host boundary ECC, and scratchpad ECC on the bit-serial register file—each independently configurable without recompilation. We also develop a complementary Monte Carlo tool for area, latency, and reliability tradeoff analysis on PIM stateful components. Evaluating nine PIMbench workloads on a BITSIMD-V analytical device, we find that scratchpad ECC granularity is the dominant overhead factor: a conservative per-logic-step charging model drives axpy to 6.8× the on-die-ECC-only baseline, while an output-only model keeps all workloads under 1.8×. These results motivate careful architectural specification of when and at what granularity in-PIM register-file protection is applied.**

I. Introduction

As processor performance gains outpace memory bandwidth, the resulting memory bottleneck increasingly limits the ability of modern systems to fully utilize advances in compute capability [1]. Processing-In-Memory (PIM) has re-emerged as a promising architectural paradigm to alleviate this bottleneck by reducing data movement between DRAM and the processor. The use cases for PIM span a wide range of domains, from resource-constrained System-on-chip (SoC) systems to high performance server-grade systems. Among these, server-grade environments are particularly compelling targets for PIM due to their reliance on cutting-edge processors and large-scale data movement. Server-grade deployments typically impose strict reliability requirements and facilitate long-running workloads, necessitating error correcting code (ECC) mechanisms to ensure data integrity over extended execution periods. However, despite the growing interest in PIM, relatively little work has explored how ECC can be effectively integrated into PIM architectures. This gap in reliability-aware PIM architectures presents a barrier to practical adoption of PIM in enterprise and datacenter environments. To address this gap, we develop a modeling framework that systematically evaluates the performance, reliability, and overhead trade-offs of deploying different ECC strategies within PIM devices, and extend the PIMeval-PIMbench simulation framework [2] to support modeling for ECC, enabling informed design decisions for future server-grade PIM architectures.

II. Background

A. Processing-in-Memory

Processing-in-memory (PIM) is an architectural paradigm which performs computation within or near memory. This technology has become particularly appealing as a mitigation for the memory wall problem, as with PIM, data movement between the host processor and main memory components is greatly reduced. Unlike conventional von Neumann systems, where data must be transferred between memory and the host processor for computation, PIM architectures embed computational capability within memory arrays or closely coupled processing elements. These processing elements can vary significantly in complexity, ranging from simple multiply–accumulate (MAC) units designed for lightweight vector or matrix operations to more general-purpose, RISC-like processing cores capable of executing broader instruction sets [3], [4]. By reducing off-chip data transfers, this approach can lower both execution latency and energy consumption.

B. Error Correcting Codes

Error Correcting Codes (ECC) are mathematical schemes that calculate redundant information for stored data bits, allowing a system to detect and potentially repair bit-flips without re-accessing the original source data. Depending on the strength of the ECC scheme employed, bit flip errors can be corrected, detected but uncorrectable, or undetectable, in which case Silent Data Corruption (SDC) can occur. While faults arising from bit-flips can be permanent (due to hardware defects or wear out) or transient (temporary flips caused by phenomena like cosmic rays), this work focuses exclusively on transient faults. We chose to focus on transient faults as they are the primary source of soft errors in sub-100nm CMOS technologies [5]. Furthermore, while ECC can temporarily mask permanent hardware defects, such faults require physical redundancy or hardware remapping for long-term remediation, which falls outside the scope of this error-model analysis.

III. Related Work

The standard implementation of ECC used in DRAM is often single-error-correction double-error-detection (SECDED), where redundant bits are employed to locate and correct single-bit errors, and detect (but not correct) double-bit errors [6]. A typical implementation of SECDED in a DRAM Dual-Inline Memory Module (DIMM) includes 8 bits of ECC to

account for 64 bits of data, requiring a 72-bit wide data path [7]. Depending on where the ECC bits are stored, DRAM ECC can fall into one of several categories. With *side-band ECC*, the ECC bits are stored in a designated DRAM chip or chips and received alongside the data by the memory controller, where SEC is performed. *Inline ECC* stores the ECC bits inside the same DRAM chip as the requested data, which is particularly useful for LPDDR, as the use of side-band ECC with the fixed channel width of LPDDR leads to wasted bandwidth [8]. *On-die ECC*, used in DDR5, similarly stores the ECC bits on the same chip as the data, however the DRAM die itself handles ECC by computing the ECC bits before a store as well as performing SEC on the ECC and data bits on a load [8].

High Bandwidth Memory (HBM) 3D-stacked DRAM technology has grown in popularity due to increased demand for AI infrastructure, which motivates new approaches to DRAM ECC due to several factors. First, the data fetch granularity of GPUs differs between GPU vendors due to variations in GPU memory hierarchy configurations. Thus, the conventional approach of DRAM ECC in assuming consistent data access size is not feasible for HBM, as an ECC method optimized for one data access size may be less effective or less energy efficient when applied to data of a different size [9]. Chen et al. [9] proposed Config-ECC, a configurable two-tier ECC scheme for HBM which reduces overall energy consumption as well as the failure in time rate compared to a fixed 64-byte DRAM ECC scheme. Through the two-tiered approach, Config-ECC can account for both small and large granularity faults across 32B, 64B, and 128B data accesses. Tier 1 corrects all small granularity faults (i.e., single-bit, single column, and single TSV) and detects large granularity faults (i.e., row, bank failure). If a large granularity fault is detected by Tier 1, Tier 2 is deployed to handle its correction.

Another issue with HBM is that the increased density of the DRAMs leads to a higher likelihood of failure, including entire entire chip failure [9], [7]. In conventional server environments, Chipkill ECC can provide coverage for entire chip failure by striping data across chips and employing a rank-wide ECC [7]. However, Chipkill is not feasible for use in HBM, as data lines are localized to individual banks within a DRAM die. To address this issue, Yoon and Erez [7] proposed a virtualized ECC mechanism which allows for the dynamic storage of ECC bits as a response to system needs [7]. Similarly to Config-ECC, this virtualized ECC is comprised of two tiers. Tier 1 consists of an extra chip that contains just enough information to perform error detection, while tier 2 contains the full correction data and is accessed only if a correction is necessary. Rather than taking up area on the DRAM chips themselves, the tier 2 information is stored elsewhere in the conventional physical address space and is handled like normal data until a correction is necessary. Gurumurthi et al. [10] show that HBM2 SECDED ECC is not sufficient for HBM3 due to increased susceptibility to cell failures as DRAM scales, and thus propose a novel on-die ECC mechanism which employs fault-isolation to reduce the uncorrected error rate of HBM3 [10].

There are several works which address ECC within the scope of analog PIM [11], [12]. To the best of our knowledge, the only work which explicitly incorporates digital PIM with DRAM ECC is Cache-PIM [6]. Cache-PIM implements a PIM-compatible last level cache inside eDRAM which is equipped with triple-tier ECC technology; however, cache-PIM is designed for analog in-memory-computing, rather than digital PIM. We are not aware of any prior work which explicitly enables in-DRAM ECC which is compatible with digital PIM.

IV. System Design

For system design, we primarily focused on building a modular, layered framework that future end-users can readily configure and extend without modifying the simulator core. Our principal contribution is a three-tier ECC hierarchy integrated into the PIMeval analytical model, mirroring the protection boundaries present in real server-grade PIM deployments: on-die DRAM ECC, memory-controller boundary ECC, and scratchpad ECC applied to the register-file words accessed during in-PIM computation. Each tier is independently configurable and can be enabled or disabled in isolation, allowing practitioners to decompose the total ECC overhead into its constituent sources. All configuration parameters are exposed both through flat key-value files compatible with PIMeval's existing `pimSimConfig` infrastructure and through environment variables, enabling automated parameter sweeps without recompilation of the simulator or any workload binary. Internally, all ECC logic is encapsulated behind abstract class interfaces, so that adding a new ECC scheme or an additional protection tier requires only the implementation of the relevant virtual methods and a registration entry in the factory. The subsections below describe the physical system model underpinning our overhead calculations, the ECC schemes implemented, and additional modeling of ECC tradeoffs on individual PIM unit components.

A. System Model

Our ECC framework targets the BITSIMD-V bit-serial PIM architecture, evaluated through PIMeval's analytical simulation mode. In BITSIMD-V, each memory subarray functions as an independent PIM core. Operand data is stored row-by-row within the subarray, and arithmetic operations are decomposed into sequences of row-level reads (n_R), writes (n_W), and bitwise logic steps (n_L), each with an associated per-operation timing parameter (t_R, t_W, t_L). The estimated runtime for a given command is therefore:

$$ms_{cmd} = n_R \cdot t_R + n_W \cdot t_W + n_L \cdot t_L$$

where the step counts for each operation and data type are determined analytically by PIMeval's bit-serial performance model. For all experiments in this work, we configure the simulated device as 1 rank $\times$ 4 banks $\times$ 4 subarrays, yielding

8 independent PIM cores, each with 1,024 rows and 1,024 columns. Experiments use 8,192 INT32 elements distributed evenly across the 8 cores, consistent with DDR5 timing parameters.

To model ECC overhead, we associate a configurable per-access latency (t_{ecc}, in nanoseconds) and energy cost (e_{ecc}, in picojoules) with each protection tier, and charge those costs based on the volume of data touched at each protection boundary during a simulation run. We define three hierarchical protection tiers:

Tier 1 — On-Die ECC (ODECC). Mandatory on all DDR5 and HBM3 devices per JEDEC specification [13], ODECC applies a 128+8 SECDED code on every DRAM row activation. Our model charges this tier for every row read or written during both PIM computation and host-device data transfers, making it an irreducible reliability floor present in all configurations.

Tier 2 — Controller ECC. Applied at the device-to-host (D2H) copy boundary, controller ECC models the SECDED protection applied by the memory controller over 64-byte data blocks as results are transferred from the PIM device to CPU memory. Because this tier fires only at copy boundaries rather than on every computation step, its overhead is fixed per D2H transfer and is zero for workloads that produce no host-side output (e.g., in-place reductions).

Tier 3 — Scratchpad ECC. During bit-serial computation, intermediate values are held in per-subarray SRAM register files. Scratchpad ECC applies SECDED protection at a 32-bit word granularity on each register-file access. We implement two charging models: a *conservative* model that charges once per logic step (proportional to n_L summed across all subarrays), reflecting a design where every intermediate partial product is protected; and an *output-only* model that charges only once per output element written, reflecting the observation that intermediate values are transient and need only the final result to be stored under ECC protection. The output-only model is motivated by the fact that register-file intermediate values are never read back by the host and are overwritten each operation cycle.

In addition to latency and energy overhead, we model expected transient error rates using a Poisson approximation parameterized by a raw bit error rate (BER). For DRAM, we adopt a BER of 10^{-10} per bit per access, representative of DDR5 before on-die correction. For SRAM scratchpad, we adopt 10^{-15} per bit per access, consistent with the extremely low soft error rates of modern on-chip SRAM [5]. At typical benchmark data volumes and runtimes, the expected number of uncorrected errors under any of our three tiers is negligibly small, confirming that SECDED provides sufficient protection for the transient-fault model considered here.

B. ECC Schemes

As noted in section III, different architectures utilize different forms of ECC, such as inline ECC and on-die ECC, to achieve optimal performance and meet their design goals. Because the hardware specifications, locations, and granularity of these ECC models varies immensely, we concluded that a "one-size-fits-all" approach would be logistically unfeasible to implement for every PIM configuration. Instead, our contribution to the PIMeval simulator contains several different built-in ECC schemes to cover a wide range of protection levels and overhead, thereby allowing the user to test each ECC scheme and determine the best baseline for their architecture. The first method, SECDED, utilizes k Hamming parity bits along with 1 overall parity bit for DED per data word, with k equaling the data width. SECDED can also be split into separate SEC and DED schemes within the simulator. The second method, CRC-32, ignores the data width, instead opting for 32 ECC bits per CRC application. While this method achieves greater performance and energy efficiency compared to SECDED, it is unable to correct any detected errors. Finally, the third scheme, Reed-Solomon, adds 8 ECC bits to every 64 bits of data. This method, like SECDED, can both correct and detect errors, and it provides a higher degree of protection against burst-errors. However, its increased protection incurs a performance cost, making it less efficient than SECDED and CRC-32.

By default, all ECC schemes utilize 128+8 on-die ECC bits, as all DDR5 and HBM3 chips must contain that number of SEC bits in accordance with JEDEC standards [13]. However, the user can configure the ECC granularity across any ECC scheme, allowing them to compare tradeoffs and select the granularity that best suits their needs. The user can also stack different granularity levels to simulate different combinations of Hamming and parity bits to further fine-tune their scheme.

Internally, all ECC schemes derive from the abstract base `pimEccStrategy` class, which contains virtual methods for encoding and decoding both uint64_t and vector data types. Each scheme also accesses scheme-specific helper methods in the `pimEcc` class. For example, the CRC-32 encode and decode functions reference a `calculateCRC32` function to compute a CRC-32 polynomial with the current data, and the respective functions for Reed-Solomon call the `computeRsCheckByte` function to determine the check byte for a 64-bit data block. When implementing custom ECC schemes, any scheme-specific helper functions should be located within this class to ensure consistent behavior.

Each ECC configuration has 5 parameters that were added to the standard PIM-Sim config files. The baseline configurations we added are located within "configs/ecc_profiles", and they can be accessed by PIMEval-PIMBench by providing the config filepath with the `pim-sim_config` argument. Each parameter can also be passed as a standalone argument if desired.

- `ecc` → Toggles ECC on/off when running the current benchmark (1 = ECC enabled, 0 = ECC disabled).
- `ecc_type` → Defines the ECC scheme to use when running the simulation.
- `ecc_granularity` → The number of parity bits utilized for ECC.
- `ecc_latency_ns` → ECC per-access performance latency in nanoseconds (ns).

- `ecc_energy_pj` → ECC energy cost in picojoules (pj).

With these parameters, the `pimEccFactory` class creates a simulated ECC Strategy object based around the provided ECC type and parameters. Because each ECC scheme is defined as a standalone class, custom ECC schemes must include a branch in the `pimEccFactory::create` method to initialize that scheme. Otherwise, if an unknown type is specified, the simulator will default to SECDED. An additional branch for the custom ECC method must also be included within `pimEcc::getTotalBitsWithECC`, as each implemented ECC method uses a different number of parity bits for each layer. The same applies for `pimEcc::encodeMultiLayer` and `pimEcc::decodeMultiLayer`, as each type currently calls a type-specific helper method for each operation.

C. ECC for Stateful Components

To understand the implications of ECC integration with respect to the unique architectural components of a PIM unit, we developed a configurable modeling tool that analyzes tradeoffs in area, latency, and reliability between various ECC schemes employed in varying types of PIM units [14]. For such, we chose to implement protection schemes for the stateful components of a PIM unit (i.e., register files, scratchpads, and/or caches), and leave the protection of the PIM computation itself to future work. The main reason we chose to do so is that most logic-level faults are filtered out by timing masking, meaning that for an error to be propagated, it must occur within a very narrow window to be captured by a latch [15]. On the other hand, stateful components in PIM units retain values, and thus are more susceptible to silent data corruption (SDC) over time, as any bit flip occurring at any time will persist until the data is next accessed or overwritten.

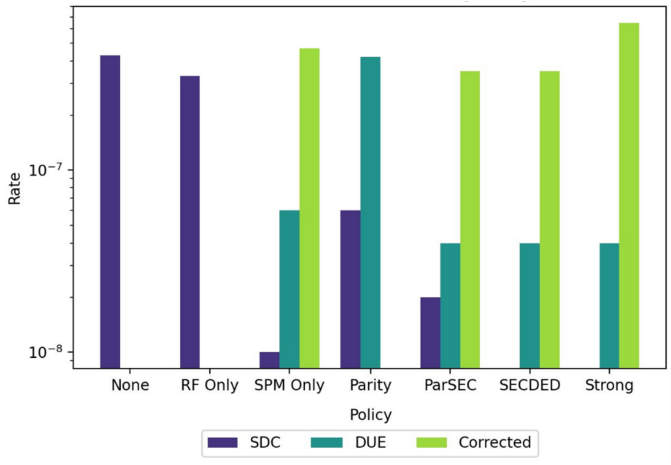

Fig. 1: Error Rates by Policy for a 1KB Register File 32KB Scratchpad PIM Unit

In order to model these tradeoffs, our framework abstracts a PIM unit as a collection of stateful components, where each PIM unit is comprised of a register file, a scratchpad, and optionally a cache memory. A stateful component maintains a size, in bits, an access rate, and a raw bit error rate (BIT). Access rates are determined by component type and capacity: register files are assigned a rate of 1.0 due to their near-constant activity being on the critical path, while larger scratchpads and caches are assigned rates between 0.1–0.3 to reflect their lower access density. The inclusion of access rate accounts for the dynamic-stress effect, where increased memory activity and read-equivalent stress have been shown to increase the occurrence of soft errors compared to static states [16]. To keep BIT consistent with the modeling in PIMeval, we adopt 10^{-15} per bit per access, scaled down to a range of 10^{-11} to 10^{-9} (depending on specific component hardening) in order to make the Monte Carlo simulations we perform computationally feasible. This acceleration allows us to observe error patterns within a practical number of simulation cycles without altering the relative reliability trends between components. The framework includes predefined ECC schemes for basic parity, SECDED, and a strong Chipkill-style protection (see Table I), but allows for the creation of new schemes. ECC schemes have an area overhead factor (normalized to the baseline, which is no ECC), a set latency penalty, the number of erroneous bits which can be detected, and the number of erroneous bits that can be corrected.

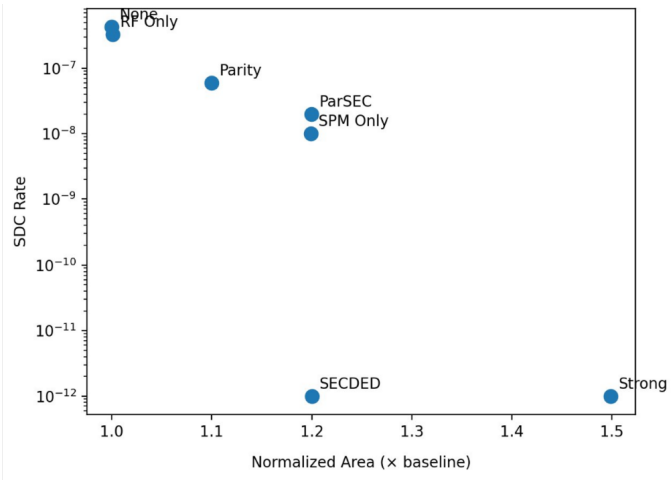

Fig. 2: Reliability vs Area Cost Tradeoff across Various ECC Schemes for a 1KB Register File 32KB Scratchpad PIM Unit

With this tool, users can input their own specifications for a pim unit and create their own ECC schemes, and then the tool will run the simulations and rates of SDC vs detected unrecoverable (DUE) vs correctable (shown in Figure 1), as well as plot the are vs reliability tradeoff against the baseline of no ECC (shown in Figure 2).

In order to assess the tradeoffs of employing different ECC schemes on a PIM unit, the framework conducts Monte Carlo simulations, which simulate randomized fault opportunities per state component to estimate corrected, DUE (detected unrecoverable), and SDC (silent data corruption) rates. Each simulation runs 1,000,000 trials. For each trial, a fault is triggered if $r < Perr$, where r is a random value sampled

TABLE I: Predefined Protection Scheme Specifications

Scheme	Area Factor	Latency Penalty (ns)	Correctable Bits	Detectable Bits
None (baseline)	1.0	0.00	0	0
Parity	1.1	0.01	0	1
SECDED	1.2	0.05	1	2
Strong	1.5	0.10	2	4

from a uniform distribution $[0, 1)$. We define the probability of an error occurring as $Perr$:

$$P_{err} = \text{size}_{\text{component}} \times \text{area overhead} \times \text{BER} \times \text{access rate}$$

When a fault is triggered, the number of affected bits is determined by sampling from a predefined soft-error distribution, where we account for single, double, and triple-bit flips. While experimental data (representing worst-case atmospheric radiation scenarios) shows that multi-bit errors can range from 9% to 17% of observed errors, our model adopts a conservative 10% estimate for multi-bit errors to avoid overestimating the failure rates of standard PIM components [16]. Overall, we adopt a distribution where 90% of faults result in single-bit flips, 8% in double-bit flips, and the remaining 2% in triple-bit flips.

V. Methodology

Since ECC modeling in PIMeval can simulate and perform ECC at several memory locations and granularities, such as the memory array, the ALU, and the scratchpad, we wanted to compare ECC performance with several diverse configurations to determine their tradeoffs. Thus, to evaluate ECC efficiency, we utilized several established PIM workloads within PIMbench, most of which utilize arithmetic and matrix operations. These benchmarks were vec-add, gemv, relu, axpy, brightness, histogram, linear-reg, prefix-sum, and select. We then ran these workloads on three different ECC modes. The first mode utilized SEC with a hamming code with on-die ECC only, which is used as a baseline against all other modes. The second mode performed SECDED on the memory controller level to simulate DRAM ECC from the memory controller level. The final mode performed SECDED as well on the scratchpad level, differing between performing error correction on arithmetic completion, or individual results. All three modes ran the benchmarks using the BITSIMD-V "analytical" mode with DDR5 hardware specifications in PIMEval-PIMBench. With this configuration, we ran all the previously mentioned benchmarks on all three ECC modes, then compared each mode's normalized total runtime against the baseline.

VI. Results

We evaluate our three-tier ECC framework across nine PIM workloads—vec-add, gemv, relu, axpy, brightness, histogram, linear-regression, prefix-sum, and select—each run on a BITSIMD-V analytical device with DDR5 timing parameters and 8,192 INT32 elements. We compare four configurations: Mode 1 (ODECC only, JEDEC baseline), Mode 2 (Mode 1 plus controller ECC on device-to-host copies), Mode 3 (Mode 2 plus scratchpad SECDED charged per logic step), and Mode 4 (Mode 2 plus scratchpad SECDED charged per output element). Figure 3 reports total runtime for each workload and mode, normalized to the Mode 1 baseline. Figure 4 shows the per-tier composition of Mode 3 total runtime.

On-Die and Controller ECC overhead is small and predictable. Because ODECC is present in all modes and is already captured by the Mode 1 baseline, its absolute cost (ranging from 0.008 ms for select to 0.068 ms for histogram) represents the mandatory reliability floor imposed by JEDEC standards rather than incremental overhead. Mode 2 adds a constant ≈0.008 ms per workload for the controller SECDED check over each device-to-host copy boundary; for workloads that produce scalar reduction results with no host-side output copy (histogram, linear-regression), this overhead is zero. The flat, workload-independent nature of both tiers makes them straightforward to account for in system-level power and performance budgets.

Mode 3 scratchpad ECC overhead is large and highly workload-dependent. When scratchpad ECC is charged once per bit-serial logic step, the resulting overhead scales with the number of logic steps required to compute each output element. Simple element-wise operations such as `pimAdd`, used by vec-add, require 97 logic steps per INT32 element and incur a Mode 3 overhead of 128% above Mode 1. By contrast, `pimScaledAdd`, used by axpy, implements a fused scalar-multiply-add via a full 32-bit shift-and-add loop requiring 2,705 logic steps per element; axpy consequently reaches 584% overhead and a Mode 3 total runtime of 6.8× Mode 1. Compound workloads involving multiple sequential operations incur proportionally larger costs: linear-regression (four reductions plus two multiplies) and histogram (sixteen equality comparisons plus sixteen reductions) reach 383% and 380% overhead, respectively. The gemv workload, despite having the largest absolute scratchpad cost (4.15 ms), shows only 104% relative overhead because its Mode 1 baseline compute time (4.95 ms) is already large, reflecting its inner BLAS-2 accumulation loop. As shown in Figure 4, scratchpad ECC accounts for between 21% and 77% of Mode 3 total runtime across workloads, confirming it as the dominant ECC cost for compute-intensive bit-serial PIM.

Mode 4 (output-only) substantially closes the gap between Mode 3 and the baseline. By charging scratchpad ECC only once per output element rather than per intermediate logic step, Mode 4 reduces overhead to between 1.2× and 1.8× Mode 1 across all nine workloads, compared to Mode 3's range of 2.0× to 6.8×. The workloads most sensitive to this modeling choice are those with the highest logic-step-to-output ratios: axpy falls from 6.8× to 1.3×, and gemv falls from 2.0× to 1.2× Mode 1 runtime. Vec-add, whose operation requires far fewer logic steps, is less affected (2.3× to 1.7×). These results suggest that the physical accuracy of the scratchpad ECC charging model has a first-order impact on projected overhead for arithmetic-heavy workloads. An output-only model, which reflects the observation that intermediate

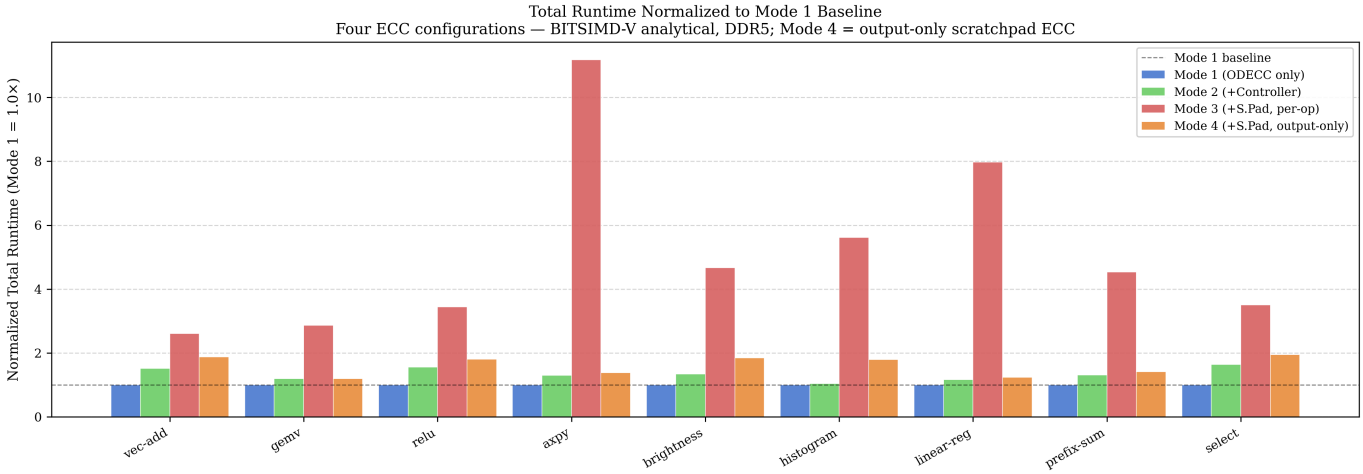

Fig. 3: Total runtime normalized to Mode 1 (ODECC-only baseline) for nine workloads across four ECC configurations. Mode 3 charges scratchpad ECC per logic step; Mode 4 charges per output element.

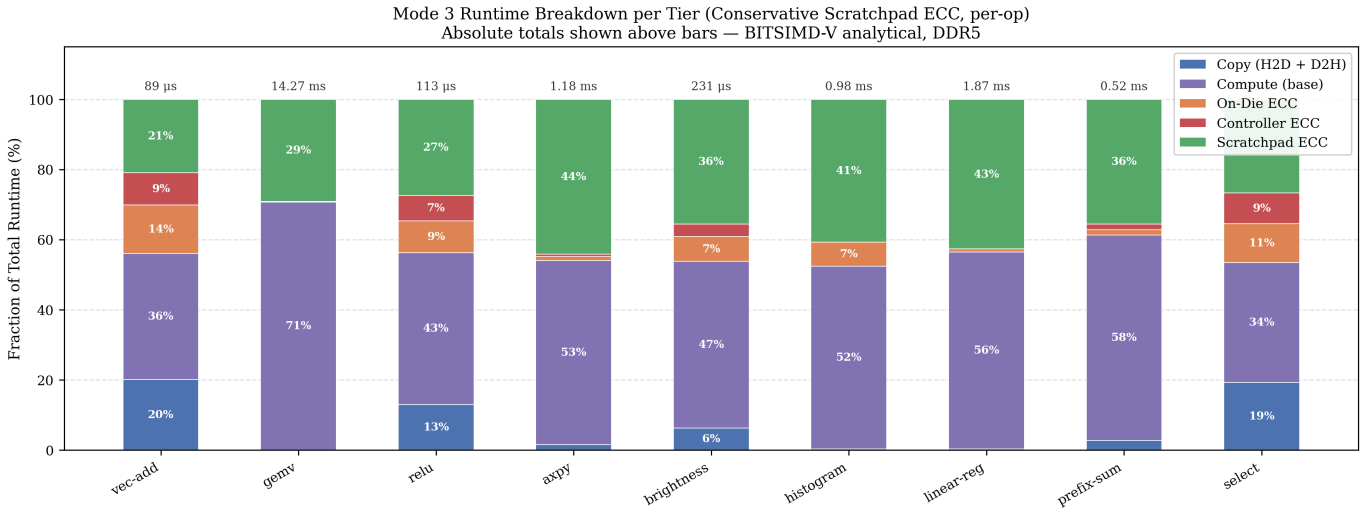

Fig. 4: Per-tier runtime breakdown for Mode 3. Absolute totals are shown above each workload.

partial products are transient and are not retained across operation boundaries, is the more conservative and physically realistic choice for general-purpose PIM performance analysis.

Taken together, these results highlight two key design implications for ECC-aware bit-serial PIM. First, the mandatory JEDEC on-die ECC tier is not negligible: it consumes up to 44% of Mode 3 total runtime on histogram, where repeated row activations for many equality comparisons amplify its cost. Second, scratchpad ECC granularity—specifically, whether protection is charged per logic step or per output element—has a larger effect on projected system performance than the choice of ECC scheme at any single tier, motivating careful architectural specification of when and at what granularity in-PIM register-file protection is applied.

VII. Future Work

Future work in this area could involve further investigation into methods for protecting the integrity of the PIM computation logic itself. Additionally, the framework for modeling ECC on the stateful components of a PIM unit could be extended to model several PIM units at once or a combination of varying PIM units, in order to provide tradeoff analyses representative of entire PIM devices, rather than just a PIM unit present in a PIM device.

VIII. Conclusion

We extended PIMeval-PIMbench with a three-tier ECC hierarchy: on-die DRAM ECC, device-to-host boundary ECC, and scratchpad ECC on the bit-serial register file. In addition, we developed a complementary Monte Carlo tool for area, latency, and reliability tradeoff analysis on PIM stateful components. Evaluation across nine PIMbench workloads shows

that scratchpad ECC granularity is the dominant cost driver: a conservative per-logic-step model pushes axpy to 6.8× the ODECC-only baseline, while an output-only model constrains all workloads to under 1.8×. The overhead model is analytical and relies on published timing parameters rather than silicon measurements, and the two sub-frameworks remain partially decoupled; addressing both limitations represents the most direct path toward a production-ready, ECC-aware PIM simulation tool.

REFERENCES

[1] N. R. Mahapatra and B. Venkatrao, "The processor-memory bottleneck: problems and solutions," *XRDS*, vol. 5, no. 3es, pp. 2–es, Apr. 1999. [Online]. Available: https://doi.org/10.1145/357783.331677

[2] U. LavaLab, "Pimeval-pimbench: Pimeval simulator and pimbench suite," https://github.com/UVA-LavaLab/PIMeval-PIMbench, 2025, accessed: 2026-04-30.

[3] J. Gómez-Luna, I. E. Hajj, I. Fernandez, C. Giannoula, G. F. Oliveira, and O. Mutlu, "Benchmarking memory-centric computing systems: Analysis of real processing-in-memory hardware," *CoRR*, vol. abs/2110.01709, 2021. [Online]. Available: https://arxiv.org/abs/2110.01709

[4] S. H. Shin, Y. Yoo, Y. Moon, N. Son, D. Kim, and S. Kang, "Star-pim: Self-test and repair structure for processing-in-memory with adder tree-based mac," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 73, no. 3, pp. 1990–2003, 2026.

[5] R. Baumann, "Soft errors in advanced computer systems," *IEEE Design & Test of Computers*, vol. 22, no. 3, pp. 258–266, 2005.

[6] S. Ha, S. Um, S. Kim, K. Sohn, and H.-J. Yoo, "Cache-pim: An ecc-compatible edram processing-in-memory for last-level cache with triple-level error correction," *IEEE Journal of Solid-State Circuits*, vol. 61, no. 2, pp. 750–762, 2026.

[7] D. H. Yoon and M. Erez, "Virtualized and flexible ecc for main memory," in *Proceedings of the 15th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '10)*. Pittsburgh, Pennsylvania, USA: ACM, 2010, pp. 371–382. [Online]. Available: https://doi.org/10.1145/1736020.1736060

[8] V. Sankaranarayanan, "Error correction code (ecc) in ddr memories," https://www.synopsys.com/articles/ecc-memory-error-correction.html, Oct. 2020, synopsys, 7 min read, accessed February 5, 2026.

[9] H.-M. Chen, S.-Y. Lee, T. Mudge, C.-J. Wu, and C. Chakrabarti, " Configurable-ECC: Architecting a Flexible ECC Scheme to Support Different Sized Accesses in High Bandwidth Memory Systems ," *IEEE Transactions on Computers*, vol. 68, no. 05, pp. 646–659, May 2019. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/TC.2018.2886884

[10] S. Gurumurthi, K. Lee, M. Jang, V. Sridharan, A. Nygren, Y. Ryu, K. Sohn, T. Kim, and H. Chung, "Hbm3 ras: Enhancing resilience at scale," *IEEE Computer Architecture Letters*, vol. 20, no. 2, pp. 158–161, 2021.

[11] D. Shi, Y. Fu, Y. Zhu, A. Fan, Y. Tao, B. Yan, and Y. Yang, "Correcting processing-in-memory multiply-accumulate arithmetic errors with LDPC," *npj Unconventional Computing*, vol. 3, no. 14, Mar. 2026. [Online]. Available: https://www.nature.com/articles/s44335-026-00014-z

[12] D. Shi, Y. Zhu, A. Fan, Y. Tao, Y. Yang, and B. Yan, "Non-Binary LDPC Arithmetic Error Correction For Processing-in-Memory," *npj Unconventional Computing*, vol. 3, no. 14, 2026. [Online]. Available: https://arxiv.org/abs/2502.11487

[13] V. Sankaranarayanan, "Error correction code (ecc) in ddr memories," *Synopsis*, 2020. [Online]. Available: https://www.synopsys.com/articles/ecc-memory-error-correction.html

[14] K. L. Farran, "PIM-ECC: A Framework for Modeling Reliability-Performance Tradeoffs in Process-in-Memory Architectures," https://github.com/klfarran/pim-ecc, 2026.

[15] M. Anglada, R. Canal, J. Aragon, and A. Gonzalez, "Maskit: soft error rate estimation for combinatorial circuits," in *IEEE International Conference on Computer Design*. Institute of Electrical and Electronics Engineers (IEEE), 10 2016, p. 614. [Online]. Available: http://ieeexplore.ieee.org/document/7753348/

[16] P. Rech, J.-M. J.-M. Galliere, P. Girard, A. Griffoni, J. Boch, F. Wrobel, F. Saigné, and L. Dilillo, "Neutron-Induced Multiple Bit Upsets on Two Commercial SRAMs Under Dynamic-Stress," *IEEE Transactions on Nuclear Science*, vol. 59, no. 4, pp. 893–899, Mar. 2012. [Online]. Available: https://hal-lirmm.ccsd.cnrs.fr/lirmm-00805031