DRAM-Compatible Deterministic B2S Conversion Using Time-Weighted Comparison and DRAM-Resident Templates

Shiyu Xia, Chen Zhang, Mingzhao Yang, Hao Meng, Mingyuan Liu, *Member, IEEE*, and Zhigang Ji, *Member, IEEE*

***Abstract*—Stochastic computing (SC) encodes numerical values as long binary bitstreams and performs arithmetic through simple bitwise logic, making it naturally suited to processing-in-memory (PIM) architectures. A key bottleneck is the conversion from compact binary numbers to these long bitstreams, known as binary-to-stochastic (B2S) conversion. When this conversion is performed outside the memory array, the resulting bitstreams must be written back into DRAM, introducing data-movement overheads that dominate end-to-end energy and latency. This work presents a deterministic in-DRAM B2S framework that performs this conversion inside DRAM banks without modifying the 1T1C cell array or sense amplifiers (SAs). The design repurposes standard SAs to perform magnitude comparisons through a time-weighted wordline-activation mechanism, and employs deterministic threshold templates stored as static DRAM rows. A three-phase pipeline coordinates initialization, RowClone-based broadcast, and weighted comparison to generate column-parallel bitstream outputs across an activated row, scalable across banks. Evaluations show that the resulting bitstreams provide statistical properties and task-level accuracy comparable to Sobol sequences, a widely used high-quality reference in SC, across the tested bitstream lengths and workloads. System-level analysis indicates that generating bitstreams directly inside DRAM avoids the bandwidth cost of moving them across the memory channel, reducing end-to-end energy and latency by more than 2× relative to external CMOS-based generators in the evaluated configurations, and providing a latency advantage over binary in-DRAM PIM at low-to-moderate precisions. These results demonstrate a practical path toward DRAM-compatible SC.**

***Index Terms*—Stochastic computing, processing-in-memory, DRAM, stochastic number generation.**

I. Introduction

STOCHASTIC computing (SC) has been extensively studied as a circuit- and system-level paradigm that enables compact arithmetic and naturally supports error-tolerant computation through stochastic bitstream representations [1], [2]. A major bottleneck in practical SC systems is the stochastic number generator (SNG), which converts n-bit binary values into unary bitstreams whose length typically scales as $\mathcal{O}(2^n)$ in conventional SC implementations to achieve acceptable accuracy.

Recent work has shown that such binary–stochastic conversions incur significant hardware cost and latency in conventional SC pipelines, primarily due to the bit-serial nature of pseudo-random number generator (PRNG)–comparator-based B2S generation, whose conversion latency grows linearly with the bitstream length [3]. Meanwhile, stochastic arithmetic naturally maps to massively parallel bitwise operations in memory arrays, making the integration of SC with in-memory computing architectures particularly attractive. However, most existing SC-in-memory approaches rely on non-volatile memories whose device-level stochastic switching or programmability can be directly exploited for stochastic representation. In DRAM-based systems, unary bitstreams are instead generated outside DRAM and written back into the array, introducing significant data-movement overhead.

Integrating B2S conversion directly into DRAM is constrained by several circuit- and system-level factors. First, existing deterministic and temporal unary SC schemes typically rely on explicit sequence control and cycle-level state updates in peripheral logic [2], [4], [5], [6], which are incompatible with DRAM's row-granular access model, sense-amplifier–driven operation, and stateless 1T1C cell array. Second, prior in-memory SC approaches predominantly exploit device-level stochasticity or multi-level programmability available in non-volatile memories [3], [7], [8], relying on stochastic switching behavior or cell-level write control that are not accessible in DRAM due to its row-granular activation, lack of per-cell programmability, and fixed access semantics. Finally, although DRAM sense amplifiers (SAs) can be reused for bulk bitwise operations via charge sharing, their native function is limited to resolving small differential signals during row activation, creating a semantic gap with the ordered magnitude comparison required for high-quality and reusable B2S conversion.

This work presents a deterministic in-DRAM SNG that performs B2S conversion within DRAM banks without modifying the cell array or SA circuitry. The key idea is to reinterpret the analog sensing behavior of standard SAs under controlled activation timing as a parallel magnitude-comparison primitive. Operand magnitudes are encoded through time-weighted wordline activation, while comparison thresholds are provided by deterministic unary sequence (DUS) templates stored as static DRAM rows and reused across conversions.

The main contributions are as follows: 1) A time-weighted, SA-based magnitude-comparison primitive that enables deterministic threshold comparison in DRAM without modifying

Received 8 January 2026; revised 17 April 2026 and 25 May 2026; accepted 29 May 2026. This article was recommended by Associate Editor L.-D. Van. *(Corresponding authors: Chen Zhang; Mingyuan Liu; Zhigang Ji.)*

Shiyu Xia, Chen Zhang, and Zhigang Ji are with the National Key Laboratory of Science and Technology on Micro/Nano Fabrication, Shanghai Jiao Tong University, Shanghai 200240, China (e-mail: chenzhang.sjtu@sjtu.edu.cn; zhigangji@sjtu.edu.cn).

Mingzhao Yang, Hao Meng, and Mingyuan Liu are with ChangXin Memory Technologies, Hefei 230088, China (e-mail: Robert.liu@cxmt.com).

Digital Object Identifier 10.1109/TCSI.2026.3701420

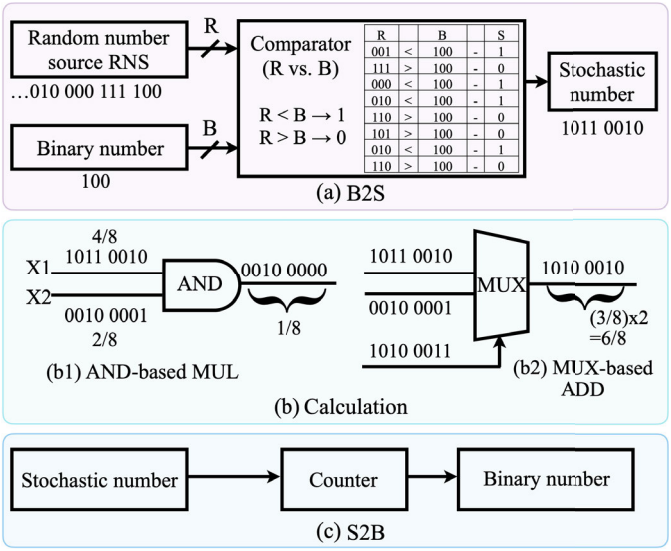


Fig. 1. Overview of SC: (a) B2S conversion using a SNG, (b) stochastic computation (e.g., multiplication using an AND gate), and (c) S2B conversion.

the 1T1C array or SA, and incurring less than 0.08% per-bank area overhead. 2) A DRAM-resident DUS template mechanism and a per-bank pipeline for column-parallel B2S conversion. 3) Comprehensive evaluations at the bitstream, operator, and task levels, demonstrating accuracy comparable to deterministic and LD baselines, over 2× energy and latency reduction relative to external SNGs, and favorable latency versus binary in-DRAM PIM at low-to-moderate precisions.

The remainder of this paper is organized as follows. Section II reviews the background and motivation. Section III presents the proposed framework. Section IV reports the evaluation results. Section V discusses scalability and system-level trade-offs. Section VI concludes the paper.

II. Preliminaries and Motivation

This section reviews SC fundamentals, deterministic and LD unary methods, and relevant DRAM architectural primitives such as RowClone [9] and TRA-based logic [10], and highlights the architectural gaps that motivate deterministic unary sequence generation in DRAM.

A. Fundamentals of Stochastic Computing

SC represents numerical values using unary bitstreams, where the probability of observing a '1' encodes the underlying real number. In the unipolar format, a value $p \in [0,1]$ is represented by the fraction of ones in the stream, whereas in the bipolar format a value $x \in [-1,1]$ is mapped as $p = (x+1)/2$. With this representation, arithmetic reduces to simple bitwise logic. For two uncorrelated unipolar streams A and B, multiplication is performed by a single AND gate, where $P(A \wedge B = 1) = P(A = 1) \cdot P(B = 1) = p_A p_B$. Addition in SC is typically implemented as a scaled operation, most commonly using a MUX-based adder. These two primitives—AND for multiplication and MUX for scaled addition—form the core arithmetic operations in classical SC.

As illustrated in Fig. 1, a typical SC datapath consists of three components: (1) a SNG that performs B2S conversion; (2) the stochastic arithmetic module, which internally includes an AND gate for multiplication and a MUX-based adder for scaled addition; and (3) a stochastic-to-binary (S2B) converter that reconstructs the numerical output.

The appeal of SC lies in its lightweight arithmetic hardware, inherent tolerance to noise and variation, and potential for massive bit-level parallelism. However, SC fundamentally trades numerical accuracy for simplicity: finite bitstream lengths introduce approximation error, and higher precision requires proportionally longer streams. Moreover, inter-stream correlation can bias SC operations, making both bitstream quality and decorrelation critical to computational accuracy [11]. Throughout this paper, the term *unary bitstream* refers to any binary sequence whose fraction of ones encodes the represented value, regardless of whether the sequence is generated randomly or deterministically. The broader term *stochastic bitstream* is used when referring to the general SC paradigm or to evaluation metrics shared across all generators.

B. Deterministic SC and Unary/Temporal Frameworks

Traditional SC relies on pseudo-random bitstreams generated by linear feedback shift registers (LFSRs) [12]. Although simple to implement, PRNGs introduce variance and correlation that limit accuracy. To mitigate these issues, recent studies have explored LD sequences [13], such as Sobol and Halton, which offer more uniform coverage and reduced finite-length error. Sobol-based generators in particular show consistently low mean absolute error (MAE) at power-of-two lengths [14], offering a stronger deterministic baseline for SC arithmetic.

A more fundamental line of work has moved toward fully deterministic unary bitstream constructions. Early deterministic SC, initiated by Jenson and Riedel [15], showed that exact stochastic arithmetic is theoretically achievable using structured unary bitstreams formed by relatively-prime lengths and fixed rotations. Despite their accuracy, these sequences grow exponentially ($\mathcal{O}(2^{2n})$) and thus do not scale to practical precisions. To overcome these limitations, later deterministic SC frameworks introduce operand downscaling with deterministic error compensation, reducing the quadratic bitstream expansion of prior methods to near-linear latency scaling [5].

Parallel efforts either introduce explicit correlation-control mechanisms or eliminate random correlation fluctuations. Schober et al. [16] proposed a unary MAC that aligns deterministic unary products through regulated timing offsets to suppress pattern correlation and enable exact accumulation within bounded input ranges. Counter-based SC architectures [17] replace randomness with a finite-state counting mechanism, yielding deterministic unary bitstreams that improve reproducibility, compactness, and accuracy while avoiding random correlation fluctuations. Building on these foundations, several deterministic computing frameworks integrate unary, temporal, or hybrid representations to support larger computational kernels such as GEMM. uGEMM [18] introduces unified unary primitives and a stability-based early termination mechanism to reduce energy. tuGEMM [6] adopts counter-driven temporal unary encoding to implement exact GEMM using an array of unary accumulators. tubGEMM [19] advances this direction with a temporal–unary/binary hybrid pipeline that substantially reduces area and energy compared to prior unary-only designs. Hybrid Temporal Computing (HTC) [20] combines rate-based bitstreams with timing-based representations, showing that deterministic temporal sequencing can reduce area and power relative to prior deterministic SC designs while preserving competitive accuracy.

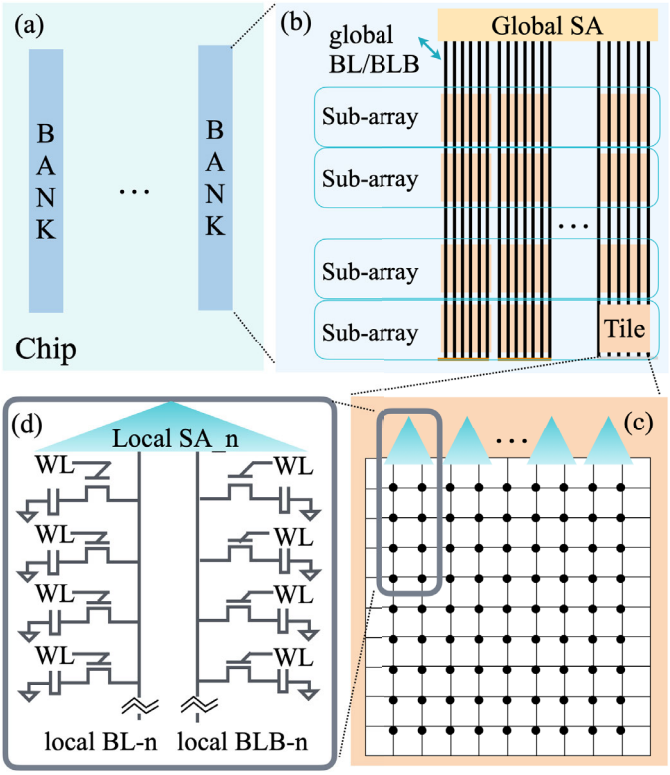


Fig. 2. DRAM hierarchical organization: (a) DRAM chip with multiple banks, (b) bank structure, (c) tile organization, and (d) local SA with cells on BL and BLB.

Several in-memory bitstream generation techniques have been explored in memristive, ReRAM, and CBRAM arrays [3], [8], [21], [22], [23], [24]. These works exploit device-level properties—probabilistic switching, tunable conductance, and in-situ logic—to perform B2S conversion or full SC flows directly within the array [23], [24]. Alam et al. [22] further demonstrate deterministic bitstream generation through in-memory logic on memristive crossbars. Deterministic LD bitstream generation has been demonstrated within memristive arrays [22], but has not yet been realized inside DRAM.

Likewise, deterministic, unary, and temporal SC schemes that operate efficiently under fully digital control typically rely on cycle-level sequencing and per-bit updates. Such assumptions are incompatible with the row-level activation model and fixed physical sensing behavior of DRAM. We therefore proceed to review the DRAM architecture and the architectural constraints it imposes on in-DRAM computation.

C. DRAM Architecture and Constraints

DRAM is organized hierarchically into banks and subarrays, as illustrated in Fig. 2. Each cell is a one-transistor–one-capacitor (1T1C) structure accessed by a wordline (WL), and each column connects to a differential bitline pair (BL/BLB) sensed by a local SA. Prior to activation, the bitlines are precharged to $V_{DD}/2$. When a WL is asserted, the cell shares its charge with the bitlines, creating a small differential voltage on the BL/BLB pair that the bistable SA resolves to full-swing logic levels (0 or 1) and restores to the cell.

Recent in-DRAM processing studies demonstrate that DRAM can support a limited set of bulk operations by leveraging its native charge-sharing behavior. RowClone [9] exploits consecutive ACTIVATE commands (Activate–Activate–Precharge, or AAP) to copy an entire row within a subarray: the first ACTIVATE loads the source row into the SAs, and the second ACTIVATE overwrites the destination cells with the row-buffer contents, enabling in-DRAM row copying without data movement over the memory channel. Ambit-style designs [10] extend this capability by issuing Triple-Row Activation (TRA), in which three cells connected to each bitline share charge. The resulting perturbation induces the SA to resolve a majority of the three inputs, enabling bulk majority behavior that can implement AND or OR by fixing one input row to all-0 or all-1.

Importantly, ComputeDRAM [25] provides experimental validation that row-copy and bitwise AND/OR operations can be realized on off-the-shelf commodity DRAM devices by exploiting timing-violating command sequences that transiently induce multi-row charge sharing. Subsequent studies have further characterized simultaneous many-row activation [26], functionally-complete Boolean logic [27], and in-DRAM true random number generation [28] in commodity DDR4 chips, confirming that a broad range of in-DRAM operations can be reliably performed through controlled timing violations.

D. Motivation and Challenges

As discussed in the previous section, commodity DRAM can support a limited set of bulk bitwise operations by exploiting charge sharing and SA behavior, making it a promising substrate for memory-centric stochastic computing.

In practice, however, the dominant bottleneck in SC systems is not the bitwise computation itself, but the generation of unary bitstreams. Converting an n-bit binary operand into an N-bit unary representation incurs significant data expansion, and N typically grows rapidly with the desired precision. When unary bitstreams are generated outside DRAM, this expansion must traverse the memory interface, introducing substantial energy and latency overhead that can outweigh the benefits of in-DRAM computation.

Achieving an in-DRAM SNG is challenging. Existing deterministic and temporal-unary computing architectures (e.g., tubGEMM [19] and HTC [20]) rely on cycle-accurate control, sequential accumulation, and fine-grained logic-side state updates, together with programmable execution scheduling. Such capabilities are fundamentally mismatched with DRAM's row-granular, sense-amplifier-driven operation model. Similarly, conventional random or LD sequence generators require multi-bit state progression and nontrivial logic (e.g., counters, Gray-code traversal, XOR networks), which cannot be embedded within the pitch-constrained 1T1C DRAM core array. Prior in-DRAM random number schemes (e.g. [29]) provide entropy but lack deterministic density and correlation control, resulting in bitstream quality well below that of LD-based approaches.

It is important to note that our objective is not to construct a new LD sequence in the classical sense. Rather, we seek a deterministic unary encoding that is compatible with DRAM's operational constraints—namely row-granular access, absence of per-bit state, and sense-amplifier-based evaluation—while providing sufficiently good statistical properties for SC. Within this constrained design space, the goal is not theoretical optimality in discrepancy, but practical effectiveness in terms of stable arithmetic behavior and controlled

correlation. This perspective reveals a fundamental gap: while DRAM efficiently supports bulk bitwise computation, it lacks a mechanism for converting stored threshold templates into column-parallel unary bitstream outputs within the array. The proposed approach addresses this gap through time-weighted row activation and sense amplification, without modifying the 1T1C cell array or SAs.

III. Proposed In-DRAM Deterministic SNG

This section presents the proposed deterministic in-DRAM SNG framework. We first overview the architecture (Section III-A), then detail the DUS construction (Section III-B), the time-weighted comparison mechanism (Section III-C), the per-bank B2S pipeline (Section III-D), and the required peripheral support (Section III-E).

A. Overview of the Proposed In-DRAM SNG Framework

The framework integrates three key components: (i) DUS templates stored as static DRAM rows, (ii) a time-weighted SA-based comparison mechanism, and (iii) a per-bank B2S pipeline that orchestrates DRAM-side execution. B2S conversion proceeds in three phases: one-time template initialization, RowClone-based operand broadcast, and time-weighted comparison. Because each bitline pair evaluates independently, a single comparison cycle produces a full-row unary bitstream whose width equals the per-bank column count (e.g., 1,024 columns in a DDR4-class bank), with further scaling across banks. The remainder of this section details each component.

B. Deterministic Unary Sequence (DUS) Construction

To enable deterministic unary bitstream generation inside DRAM without recursive logic or per-bit sequencing, we propose a DUS construction that explicitly decouples unary value correctness from bit-position distribution. DUS consists of two complementary DRAM-resident threshold templates: an Ascending Deterministic Unary Sequence (ADUS) that guarantees exact unary density, and a Shuffled Deterministic Unary Sequence (SDUS) that redistributes this density to improve spatial uniformity and inter-stream independence.

For an n-bit operand precision with $N = 2^n$ positions in the unary bitstream, ADUS stores the strictly ordered threshold set $\mathrm{ADUS}(i) = i$ for $i \in \{0, \ldots, N-1\}$ across DRAM columns. Under the binary threshold comparison rule $\mathrm{bit}[i] = (M > \mathrm{ADUS}(i))$, an operand M produces a deterministic unary prefix of M ones followed by zeros. Because each threshold value appears exactly once and in monotonically increasing order, the total number of ones exactly matches the input magnitude, enabling an exact and deterministic B2S conversion. The construction generalizes to any precision: the sequence length $N = 2^n$ and the number of template rows (n per sequence) scale with n, while the per-conversion latency grows linearly with n, not with N. The total per-bank template storage is $2n + 2$ rows (ADUS, SDUS, and two constant rows), which remains fixed regardless of workload size.

While ADUS preserves exact unary density, its monotonic structure clusters all one-positions at the beginning of the bitstream, which can induce strong correlation when multiple streams interact. SDUS mitigates this effect by redistributing the same threshold set through a deterministic full-period permutation. Specifically, let a be an integer coprime with N; the permuted thresholds are defined as $\mathrm{SDUS}(i) = (a \cdot i) \bmod N$. This mapping forms a bijection over $\{0, \ldots, N-1\}$, preserving the exact threshold set of ADUS while redistributing one-positions across the entire bitstream. Under the same comparison rule, the resulting bitstream retains the exact unary density M/N while avoiding a contiguous unary prefix. The scrambling factor a is selected offline through a reproducible procedure: for each target length $N = 2^n$, all $N/2$ odd integers in $[1, N-1]$ (coprime with N) are enumerated, and the value minimizing the star discrepancy

$$D_N^* = \sup_{\mathbf{u} \in [0,1)^2} \left| \frac{|\{j : \mathbf{x}_j \in [\mathbf{0}, \mathbf{u})\}|}{N} - u_1 u_2 \right| \tag{1}$$

of the normalized point set $\mathbf{x}_j = (j/N, (a \cdot j \bmod N)/N)$, $j = 0, \ldots, N-1$, is selected [30]. Geometrically, D_N^* measures the worst-case deviation between the empirical and uniform distributions over all axis-aligned rectangles anchored at the origin. This criterion ensures uniform joint coverage of the two threshold dimensions, which directly governs pairwise bitstream independence and is independent of any downstream accuracy metric. This construction corresponds to a rank-1 lattice rule [31], [32]. For the evaluated lengths, the selected values are $a = 7, 15, 29, 75, 95, 215, 447$ for $N = 16, 32, 64, 128, 256, 512, 1{,}024$, respectively. Once determined, a is fixed and stored as a static DRAM row, requiring no runtime logic. The SDUS permutation shares the same design principle as Van der Corput (VDC) sequences: both deterministically redistribute a monotone counter to achieve uniform coverage, with VDC applying bit-reversal and SDUS applying a multiplicative modular mapping. Either sequence, as well as Sobol, can be stored as static DRAM rows within the proposed pipeline; DUS is adopted here for its single-parameter transparency.

As shown in Sections IV-B and IV-C, the combined ADUS–SDUS construction achieves bitstream independence and arithmetic accuracy comparable to Sobol across practical lengths ($N = 16$–$1{,}024$).

C. Time-Weighted Comparison Inside DRAM

Unlike prior works that modify SAs or introduce additional analog support [33], [34], the proposed design preserves the native SA and exploits only its intrinsic charge-sharing and sensing behavior under time-weighted wordline activation. An n-bit operand and an n-bit threshold are encoded on the BL and BLB sides using wordline activations whose durations scale with bit significance. A sequence of controlled `ACTIVATE` pulses accumulates weighted perturbations on BL and BLB, and a single final sensing step resolves the resulting BL/BLB differential into a deterministic unary bit.

This behavior is governed by the bitline's first-order RC dynamics: when a wordline is enabled, the cell capacitor C_{cell} shares charge with the much larger bitline capacitance C_{BL}, producing a charge-sharing voltage $V_{\mathrm{CS}}(t)$ whose magnitude increases monotonically with the wordline-on duration. A longer activation window (e.g., t_{bit1}) therefore pushes the bitline further from its initial level than a shorter window (e.g., t_{bit8}), providing a natural mechanism for encoding per-bit weights in the time domain.

As illustrated in Fig. 3, an 8-bit operand is mapped to eight rows on the BL side (WL1–WL8). These rows are activated sequentially in descending weight order, from t_{bit1} to t_{bit8}, so that higher-order bits apply longer—and therefore stronger—charge-sharing intervals to the bitline. Sequential

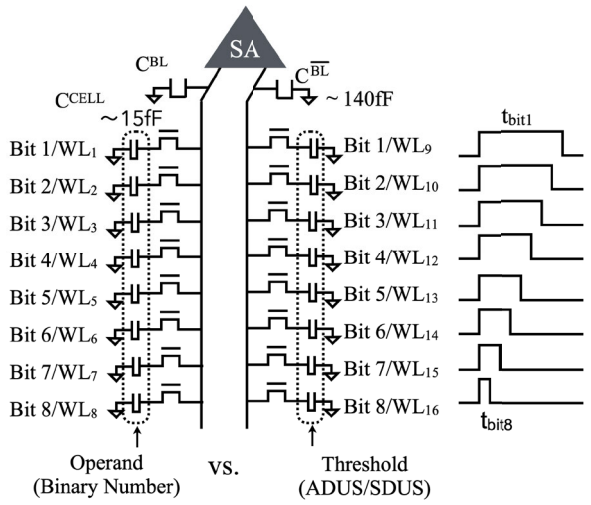


Fig. 3. Time-weighted in-DRAM comparator using sequential wordline activations for operand and threshold rows, with final sensing performed once after all pulses.

activation avoids the surge current of simultaneous multi-row activation and preserves the accumulated charge-sharing state. To maintain differential symmetry, BL and BLB activations proceed in an interleaved, weight-matched pair: for each weight level, the BL operand row is activated first (e.g., WL1), immediately followed by the corresponding BLB threshold row (e.g., WL9). The sequence continues in descending weight order (WL2–WL10,..., WL8–WL16), preventing either side from dominating the accumulated differential before the final sensing operation.

The comparison threshold is encoded on the BLB side using the DUS (ADUS/SDUS) template, with each threshold bit applied through its corresponding activation window. After both sides complete their interleaved sequences, the SA is triggered once to resolve the accumulated BL/BLB differential: if the operand's weighted charge exceeds the DUS threshold, the SA drives BL high ("1"); otherwise low ("0"). Each such evaluation produces one bit of the deterministic unary bitstream.

Importantly, PRADA [35] shows that multi-step WL activation sequences—where several rows are enabled sequentially and sensing occurs only once at the end—can construct higher-order charge-sharing operations such as 5RA and 10RA, confirming that controlled, staged charge sharing is compatible with DRAM operating margins and supporting the physical feasibility of the proposed time-weighted comparator.

We validate this mechanism using SPICE simulations with a TSMC 65-nm PDK. The model uses C_{cell}=15 fF, C_{BL}=140 fF, rise/fall times of 0.1 ns, and weighted wordline durations ranging from 1 ns to 8 ns. Three test scenarios were evaluated to assess the SA's accuracy, where the values in BL and BLB are very close, differing by only 1 bit:

1. *Near-Minimum Range:* BLB holds "0000 0001" while BL stores "0000 0010" or "0000 0000".
2. *Mid-Range:* BLB holds "0101 0101" while BL stores "0101 0110" or "0101 0100".
3. *Near-Maximum Range:* BLB holds "1111 1110" while BL stores "1111 1111" or "1111 1101". As shown in Fig. 4, the SA reliably resolves such minimal differences across the entire value range, confirming the robustness of the time-weighted comparison mechanism.

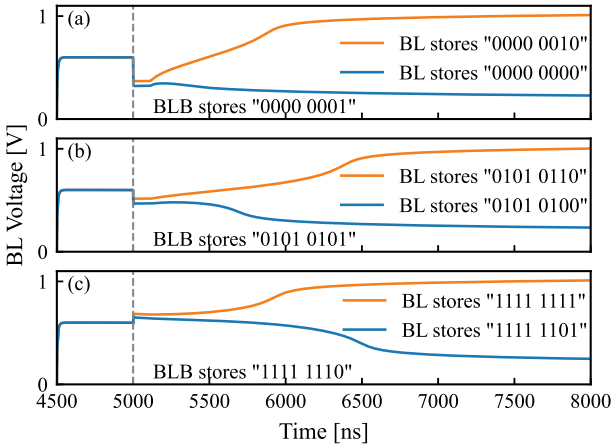


Fig. 4. SPICE validation of the proposed time-weighted comparator under near-minimum, mid-range, and near-maximum operand values.

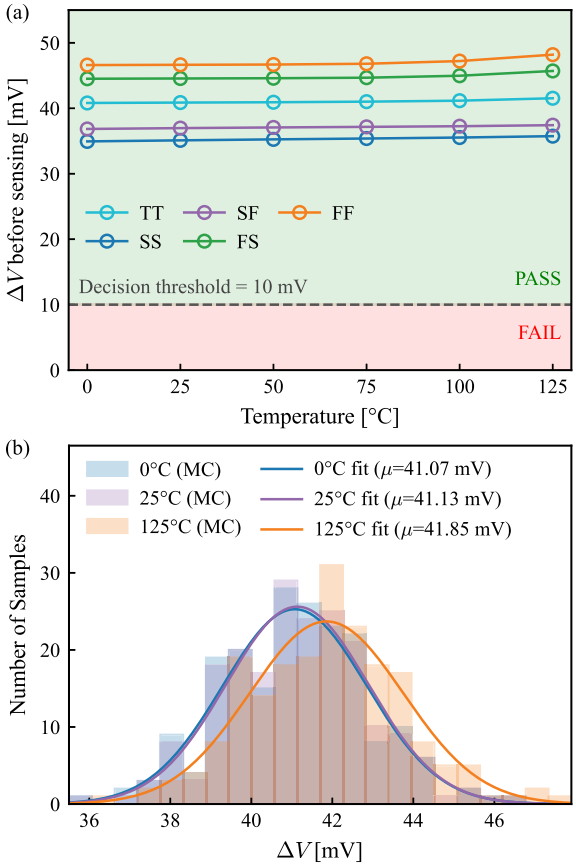


Fig. 5. Robustness of the time-weighted comparator under process and temperature variation: (a) corner analysis across five process corners and seven temperatures, and (b) 200-run Monte Carlo distributions at 0, 25, and 125 °C.

To assess robustness under process and temperature variation, we further performed corner and Monte Carlo simulations, with results summarized in Fig. 5. Across five process corners (TT, SS, FF, SF, FS) and operating temperatures spanning 0–125 °C, the minimum observed pre-sensing differential voltage is 35 mV (SS corner, 0 °C), providing approximately 3.5× margin over a representative 10 mV SA decision threshold, consistent with offset voltage levels reported for latch-type

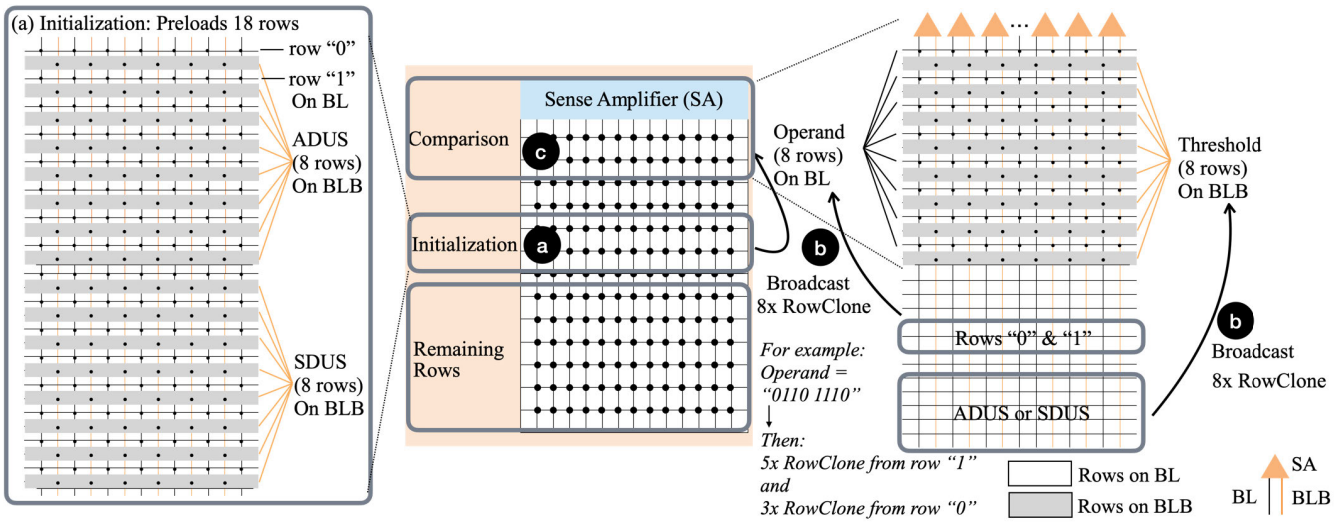


Fig. 6. Three-phase in-DRAM deterministic B2S conversion: (a) Initialization (Preload), (b) Broadcast (RowClone), and (c) Comparison (Time-Weighted).

DRAM SAs at scaled nodes [36]. A 200-run Monte Carlo analysis at 0, 25, and 125 °C yields a mean ΔV tightly clustered around 41 mV with no sample falling below this threshold across 600 trials. Cell leakage at elevated temperatures imposes no additional constraint, since the entire B2S comparison sequence completes in under 700 ns, more than $10^4\times$ shorter than the standard JEDEC refresh interval.

D. Per-Bank Pipeline for In-DRAM Deterministic B2S

Fig. 6 summarizes the three-phase pipeline that converts an n-bit binary operand into a deterministic unary bitstream within each DRAM bank. The additional support is a per-bank timing unit that generates the fine-grained WL-on durations required during the time-weighted comparison stage.

The first phase is a one-time initialization performed before any conversion begins. As illustrated in Fig. 6(a), each bank is preloaded with the static templates required by the framework: the n-row ADUS template, the n-row SDUS template, and two constant rows storing all zeros and all ones. These rows are initialized using standard `WRITE` commands and their logical contents persist across the workload, as normal DRAM refresh preserves the stored values. For n=8 with 256 threshold values, the templates may optionally be expanded horizontally to fill the 1,024-column width using relatively-prime length extension, clock-division scheduling, or rotation-based replication [15], [37], improving effective precision at no runtime cost since all columns evaluate in parallel.

In the second Broadcast phase, the bank constructs the operand and threshold representations by using RowClone to broadcast their bit patterns into the comparison zone. As illustrated in Fig. 6(b), each bit of the operand is materialized by a RowClone copy of either the all-zero row or the all-one row into the corresponding BL-side row, producing a bit-plane encoding of the operand for subsequent unary bitstream generation through eight RowClone operations. A second set of eight RowClone operations populates the BLB-side rows with the selected threshold template (ADUS or SDUS). After these broadcasts, each bitline pair holds two aligned values—the operand encoding on BL and the deterministic threshold on BLB. All data movement in this phase is performed entirely through RowClone, avoiding explicit write commands and thereby reducing both energy consumption and latency.

In the final Comparison phase, the operand and threshold rows prepared in the previous steps are compared using the time-weighted mechanism introduced in Section III-C. Each bit position i is driven with its assigned wordline-on duration t_i, and the BL and BLB rows are activated in the prescribed sequence to accumulate their respective weighted charges. After all pulses have been applied, a single sensing operation resolves the BL/BLB differential, producing one output bit per bitline pair. Because all bitline pairs evaluate in parallel, this yields a full-width unary bitstream output (1,024 bits in the modeled DRAM bank) within a single comparison cycle. Since the comparison result resides on the BL side, issuing a final `ACTIVATE` to a designated destination row naturally restores the unary bitstream output into that row's cells. Multiple banks may operate concurrently to extend the effective unary bitstream length or to instantiate multiple SNGs.

Together, these three phases provide a fully DRAM-compatible B2S pipeline that operates without modifying the cell array or SAs, while exploiting full bitline and bank-level parallelism for high-throughput unary bitstream generation.

E. Hardware Support for In-DRAM B2S Conversion

The proposed in-DRAM deterministic B2S conversion relies on a multi-step, time-weighted comparison process that requires explicit control over wordline enable duration and SA triggering. Such behavior is not supported by the activation micro-architecture of commodity DRAM, where WL activation and SA sensing are tightly coupled with fixed internal timing. Enabling deterministic, multi-step comparison therefore necessitates targeted extensions to controller sequencing and bank-local activation control, which are confined to digital control and timing logic in the memory controller and DRAM bank periphery. Fig. 7 illustrates the resulting architecture, where four proposed additions (blue blocks) augment the otherwise unmodified DRAM components (gray blocks).

On the controller side, a B2S micro-sequencer (① in Fig. 7) is integrated into the command scheduler. It contains configuration registers and a counter FSM that expand each B2S conversion into a predefined sequence of activation and

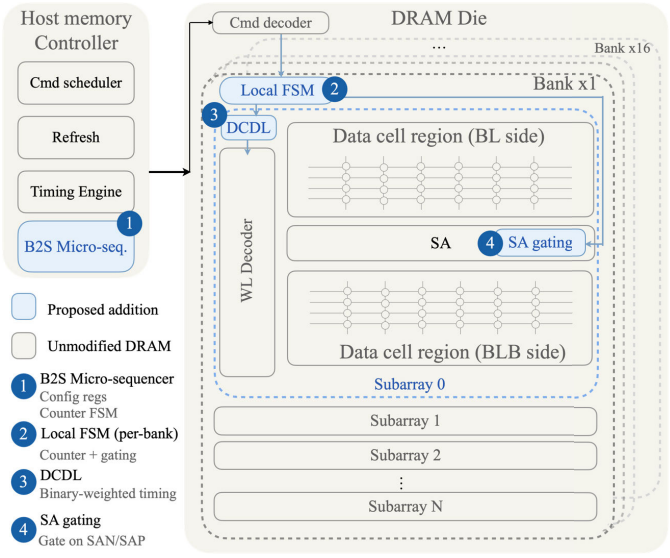


Fig. 7. Architecture overview of the proposed B2S hardware support. Blue blocks (①–④) denote proposed additions; gray blocks are unmodified DRAM components.

precharge operations while programming the corresponding bank-local WL timing parameters.

Within each DRAM bank, three compact blocks augment the existing digital periphery. A local FSM (②) manages step sequencing and coordinates the bank-side execution phases. A digitally controlled delay line (DCDL, ③) synthesizes binary-weighted WL-on durations t_{WL} by refining coarse timing windows derived from DLL-generated multi-phase clocks. An SA gating block (④) suppresses SA sensing during the first $N-1$ charge-sharing steps and enables sensing and restoration only in the final step to resolve the accumulated BL/BLB differential.

This multi-step execution is managed locally and does not alter the SA circuitry or cell-array organization, nor does it introduce additional WL activations or sensing operations beyond those required by the B2S sequence itself. All four components are implemented using standard digital building blocks and are transparent during normal DRAM operation.

We synthesized the proposed control logic using Synopsys Design Compiler with a TSMC 65-nm standard-cell library. The off-chip B2S micro-sequencer occupies 1478 μm^2 ($\approx$ 1,242 gate equivalents, 226.9 μW), while the per-bank on-chip logic totals 731 μm^2 ($\approx$ 640 gate equivalents, 230.3 μW). Given that a DDR4 bank typically occupies 1–2 mm^2 in a comparable node, the per-bank overhead is less than 0.08% of the total bank area.

IV. Evaluation and Results

This section evaluates the proposed in-DRAM SNG framework at the bitstream, operator, and task levels. We assess bitstream quality and arithmetic accuracy, and quantify latency and energy using Ramulator [38] and DRAMPower [39]. We further evaluate image-level behavior using a 3×3 Sobel operator and a full Canny pipeline on the BSDS500 dataset.

A. Experimental Setup

All in-DRAM experiments are evaluated using Ramulator and DRAMPower under a DDR4-2400R timing configuration (tCK = 0.833 ns); DRAMPower estimates energy from IDD-based command-level models per the JEDEC specification, independent of any fabrication process node. The simulated system includes one channel and one rank, each rank containing 16 banks, with 32,768 rows and 1,024 columns per bank. Key timing and organizational parameters are listed in Table I. These settings are used consistently across all methods to ensure a comparable evaluation environment.

TABLE I
DRAM Configuration Used in Ramulator and DRAMPower

Hierarchy / Organization		Key Timing Parameters	
Channel	1	t_{CK}	0.833 ns
Ranks / Channel	1	t_{CL} / t_{RCD} / t_{RP}	16 / 16 / 16
Banks / Rank	16	t_{RAS}	39
Rows per Bank	32,768	t_{RC}	55
Columns per Row	1,024	t_{CWL}	12
Speed Bin	DDR4-2400R	t_{RFC} / t_{REFI}	312 / 9360

We benchmark the proposed DUS generator against a broad set of stochastic-computing baselines spanning pseudo-random, low-discrepancy (LD), deterministic, and unary–temporal paradigms. Random denotes an idealized stochastic number stream in which each threshold is sampled i.i.d. uniformly over $[0, N)$, independently for each SNG lane and each trial, modeling a TRNG-driven SNG at the behavioral level following the standard SC evaluation protocol [11]. This baseline serves solely as a statistical reference and does not correspond to a hardware-realizable SNG. LFSR uses a single maximal-length polynomial shared by both SNGs, with the second SNG seeded at the optimal offset Δ_{idx} for each bit-precision Q (e.g. $x^8+x^6+x^5+x^4+1$, Δ_{idx}=97 at Q=8), following the optimally-seeded single-polynomial configuration of [11]. For the LD baselines, Halton uses the classical two-dimensional sequence with prime bases $(2, 3)$ [40]; Sobol uses the first two dimensions of a Joe–Kuo construction [14]; and VDC adopts the Powers-of-2 Van der Corput pairing (dim A: VDC base-2; dim B: VDC base-N) proposed by Moghadam et al. [41]. All LD sequences are unscrambled. Several deterministic and unary–temporal architectures are also included—namely Downscale [5], HTC [20], uGEMM [18], and tubGEMM [19]. Although these rely on cycle-accurate CMOS timing logic and cannot be implemented within a DRAM array, they provide meaningful deterministic baselines for evaluating accuracy. For bitstream-level assessments, DUS uses fixed ADUS/SDUS rows to reflect its static in-DRAM template model. LD baselines are evaluated under both fresh-per-trial and fixed-sequence protocols, both yielding consistent results across all reported metrics.

Bitstream-level sequence quality is evaluated using SCC [42] and ZCE [43], two widely used metrics for characterizing bitstream independence. All methods are assessed over 10,000 Monte Carlo trials with bitstream lengths N = 16–1,024, using identical unipolar mappings and operators (bitwise AND for multiplication and MUX-based scaled addition) to ensure fairness across baselines. For all MAE and correlation measurements, input operands are sampled independently from a uniform distribution over $[0, 1]$, and the same operand samples are reused across all SNG baselines.

Latency and energy are obtained by combining cycle-accurate traces from Ramulator with DRAMPower's command-level energy estimates. The B2S flow is mapped

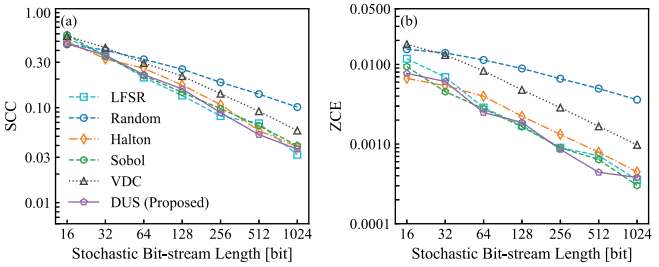


Fig. 8. Correlation metrics for stochastic bitstream pairs at varying stream lengths N: (a) SCC and (b) ZCE.

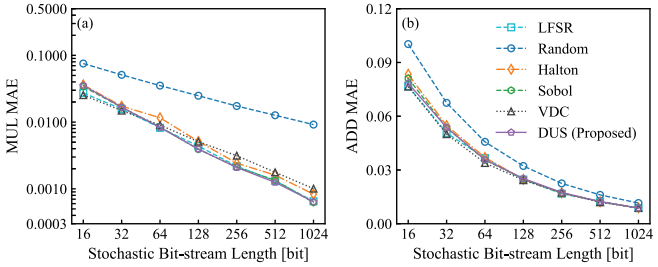


Fig. 9. MAE of unary arithmetic at varying stochastic bitstream lengths N: (a) multiplication (MUL) and (b) addition (ADD).

directly to DRAM commands following the initialization, broadcast, and time-weighted comparison phases described earlier. A conservative modeling choice is adopted for the comparison phase: each weighted activation is emulated using a standard `ACTIVATE` that incurs the full t_{RCD} timing window, yielding upper-bound latency and energy estimates. DRAMPower captures the energy of array-level DRAM commands. The additional digital control logic dissipates 230.3 μW (per-bank) and 226.9 μW (micro-sequencer) during B2S execution, contributing approximately 0.98 nJ per batch in total, which is less than 0.12% of the array-level batch energy (0.851 μJ) and does not materially affect the end-to-end comparisons.

Operator- and task-level accuracy is evaluated using a 3×3 Sobel edge operator and a full Canny detector on the BSDS500 dataset. The Sobel operator is widely used in SC-based image-processing pipelines [44]. The Canny detector [45] and the BSDS500 dataset [46] remain standard baselines for evaluating edge detection in stochastic and approximate vision pipelines. All methods operate under a unified effective unary bitstream length of N=256. Only the gradient computation stage is implemented stochastically, while all subsequent Canny processing stages use conventional fixed-point arithmetic. This setup isolates the numerical impact of stochastic Sobel-gradient magnitude computation, while maintaining a consistent vision pipeline across baselines. Accuracy metrics (MAE, ODS/OIS F-scores, and average precision) are reported in the corresponding result sections.

B. Bitstream Quality

The reliability of stochastic computing depends critically on the statistical quality and pairwise independence of the underlying unary bitstreams. We evaluate these properties for the proposed DUS sequences using two widely adopted metrics—SCC and ZCE—and compare them against four representative sequence generators: Random, LFSR, Halton, and Sobol. All experiments use the unified Monte Carlo configuration described in Section IV-A, with bitstream lengths ranging from N = 16 to N = 1,024.

SCC measures linear dependence between paired unary bitstreams and is the standard indicator of correlation-related error in stochastic arithmetic. As shown in Fig. 8, Random and LFSR sequences exhibit elevated SCC at short lengths due to clustering and cycle structure, gradually trending toward lower correlation as N increases. Halton and Sobol maintain consistently low SCC for $N \geq 64$, reflecting their well-dispersed threshold positions. The proposed DUS achieves SCC values comparable to those of Sobol across all tested lengths. For example, at N = 16, DUS yields 0.357 compared with Sobol's 0.347, and both converge to approximately 0.04 at N = 1,024. Across the full range, DUS maintains reliably low pairwise correlation.

ZCE provides a complementary measure of finite-length independence by capturing deviations from ideal zero-correlation behavior in the alignment of paired bitstreams. As shown in Fig. 8, DUS again demonstrates correlation levels comparable to Sobol. At N = 128, DUS achieves a ZCE of 0.0019 versus Sobol's 0.0016, and both fall below 5×10^{-4} by N = 1,024. In contrast, LFSR converges more slowly, and fully random sequences exhibit higher variance at short lengths.

Taken together, the SCC and ZCE results indicate that, for unary bitstream lengths between N = 16 and N = 1,024, the proposed DUS method provides reliable independence properties that scale favorably with bitstream length. While DUS does not rely on classical Sobol-style LD constructions (e.g., direction numbers or Gray-code traversal), the ADUS–SDUS combination yields cross-stream correlation metrics comparable to Sobol over the tested range, with LFSR converging more slowly at short lengths. These statistical characteristics provide a solid foundation for the arithmetic accuracy results presented in the next section.

C. Arithmetic Accuracy

We evaluate the arithmetic accuracy of stochastic computing by measuring the MAE of two fundamental unary operations: multiplication (MUL) and scaled addition (ADD).

Fig. 9(a) reports the MUL MAE results. Random and LFSR sequences exhibit higher error at short lengths due to correlation artifacts and uneven local density. In contrast, Halton and Sobol achieve lower error as a result of their more evenly dispersed threshold positions. Across the evaluated bitstream lengths, the MUL MAE of ADUS×SDUS remains on par with Sobol. For example, at N = 256, DUS yields an MAE of 0.00210 compared with Sobol's 0.00218, and both converge to the same low-error regime at N = 1,024 (approximately 6.4×10^{-4} for DUS and 6.3×10^{-4} for Sobol). Fig. 9(b) also summarizes the ADD MAE results. Because stochastic addition is inherently less sensitive to bitstream-to-bitstream correlation than multiplication, the differences between generators narrow. Halton and Sobol maintain low error across all lengths, and DUS again matches their performance. At N = 512, DUS achieves an MAE of 0.02476, comparable to Sobol's 0.02483.

Overall, these results show that, under identical unary operators and operand distributions, the proposed DUS delivers multiplication and addition MAE that are empirically indistinguishable from Sobol across the tested bitstream-length range.

TABLE II

LATENCY AND ENERGY BREAKDOWN OF ONE FULL B2S BATCH

Phase	#Rows	Cycles	Time (ns)	Energy (μJ)
(a) Initialization	18	6761	5631.9	4.170
(b) Broadcast	16	1706	1421.1	0.554
(c) Comparison	16	834	694.7	0.297

Initialization is a one-time cost amortized across batches.

In other words, for practical unary SC kernels and $N \leq 1{,}024$, DUS matches the arithmetic accuracy of well-established LD generators while avoiding their recursive logic and direction-number hardware.

D. Energy and Latency

This subsection evaluates the latency and energy overhead of the proposed in-DRAM deterministic unary bitstream generation, with a focus on the B2S conversion pipeline. All measurements are obtained under the same DRAM configuration described in Section IV-A, ensuring a consistent comparison across methods.

As introduced in Section III-D, the B2S pipeline consists of three DRAM-side phases: initialization, operand broadcast, and time-weighted comparison. Each phase contributes differently to the overall cost. Table II summarizes the detailed latency and energy breakdown for generating one unary batch, where a batch produces 16,384 unary bits in total (1,024 columns per bank across 16 banks).

The initialization phase preloads 18 template rows per bank (8 ADUS, 8 SDUS, and 2 constant rows) at a one-time device-level cost of 5.63 μs and 4.17 μJ. These templates are preserved by standard refresh and reused across all subsequent batches; for a 256×256 GEMM workload, the amortized initialization overhead is below 0.1% of the total pipeline cost. The broadcast phase uses 16 RowClone operations per bank (8 for operand, 8 for threshold template). The comparison phase executes the time-weighted comparison described in Section III-C.

Excluding the amortized initialization cost, the combined broadcast and comparison phases require 2116 ns and 0.851 μJ to generate one full 16,384-bit unary batch. This corresponds to an average latency of 0.129 ns per bit and an effective throughput of 7.74 bits/ns when activations are distributed across 16 banks. This per-batch cost is independent of how the 16,384 bits are partitioned: a single row can host one 1,024-bit stream, four 256-bit streams, or sixteen 64-bit streams per bank at the same cost. Since external SNG generation and transfer costs also scale linearly with N, the in-DRAM advantage remains essentially invariant across bitstream lengths.

The reported latency and energy costs correspond to the proposed B2S pipeline with a fixed per-bank row allocation. Under a standard DDR4-class configuration with 32,768 rows per bank, the pipeline reuses 34 DRAM rows (0.1% of bank capacity), independent of the target unary bitstream length.

E. Operator-Level and Task-Level Evaluations

We evaluate the numerical accuracy of the proposed deterministic in-DRAM bitstreams at both the operator and task levels using two representative vision settings: (i) a 3×3 Sobel operator, which reflects local stencil-level arithmetic behavior, and (ii) a full Canny detector on the BSDS500 dataset, which assesses stability in an end-to-end vision pipeline. All methods operate under the unified configuration defined in Section IV-A.

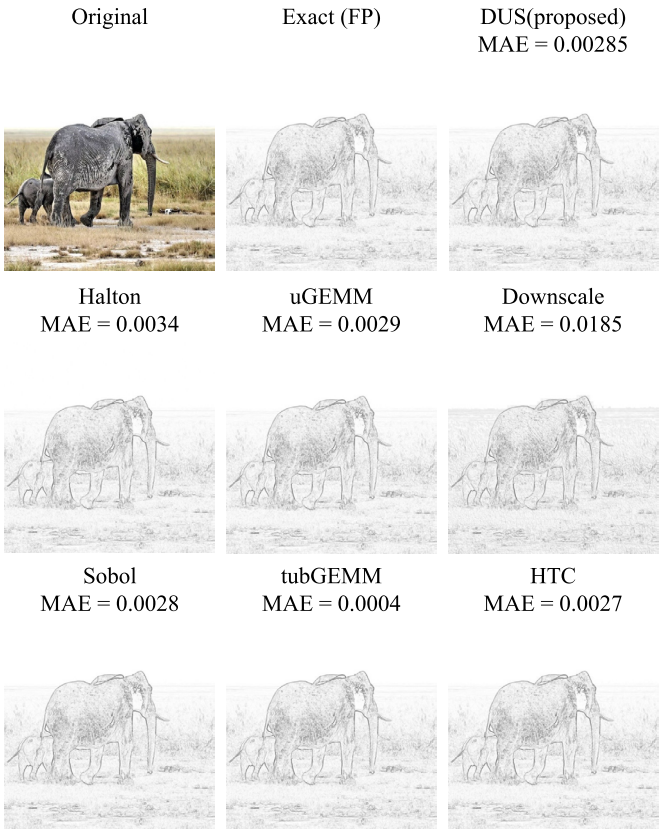


Fig. 10. Sobel edge detection results with the Exact (FP) baseline, the proposed DUS method, and other SC methods (Halton, Sobol, uGEMM [18], tubGEMM [19], Downscale [5], HTC [20]). Reported MAE values are measured against the Exact (FP) baseline.

TABLE III

BSDS500 EVALUATION UNDER A UNIFIED CANNY PIPELINE. ALL METHODS USE AN EQUIVALENT UNARY BITSTREAM LENGTH OF N=256

Method	Class	ODS (F)	OIS (F)	AP
Exact (FP baseline)	–	0.523	0.544	0.436
DUS (proposed)	DRAM-compatible	0.521	0.543	0.442
HTC [20]	Off-array CMOS	0.522	0.545	0.438
uGEMM [18]	Off-array CMOS	0.422	0.453	0.355
tubGEMM [19]	Off-array CMOS	0.522	0.543	0.435
Downscale [5]	Off-array CMOS	0.515	0.541	0.486
Sobol	Off-array CMOS	0.523	0.547	0.441
Halton	Off-array CMOS	0.519	0.542	0.452
VDC	Off-array CMOS	0.522	0.546	0.439

At the operator level, Fig. 10 compares Sobel edge maps generated by the proposed DUS an several representative stochastic computing baselines. The DUS result closely follows the floating-point reference and achieves a Sobel gradient magnitude MAE of 2.85×10^{-3}, comparable to Sobol (2.8×10^{-3}) and uGEMM (2.9×10^{-3}). tubGEMM attains a lower MAE ($\approx 4\times10^{-4}$) due to its exact temporal accumulation, which relies on sequential digital operations that are not compatible with DRAM execution constraints. These results indicate that DUS provides operator-level accuracy comparable to established deterministic and LD baselines while remaining DRAM-compatible.

At the task level, each stochastic gradient estimator is integrated into the same Canny evaluation flow. As summarized in Table III, the proposed DUS achieves an ODS F-score of 0.521, closely matching Sobol (0.523), tubGEMM (0.522), and the floating-point reference (0.523), with similar precision–recall characteristics.

uGEMM exhibits a reduced ODS score (0.422). Although uGEMM performs well in isolated Sobel filtering, its pixel-conditioned unary gating advances the weight stream only when the pixel stream fires. On BSDS500—where pixel statistics vary substantially across textures and edge strengths—this asymmetric update rule reduces the effective gradient dynamic range, leading to the degraded PR performance observed in the full Canny pipeline.

Overall, the deterministic structure of the proposed DUS bitstreams preserves gradient ordering and variance consistently across images, resulting in operator- and task-level accuracy comparable to deterministic and LD baselines.

V. Discussion

This section examines the system-level behavior of the proposed in-DRAM deterministic SNG. We analyze scalability across the DRAM hierarchy and compare in-DRAM unary bitstream generation with external SNGs and binary in-DRAM PIM in terms of end-to-end energy, latency, and precision trade-offs. We further discuss practical considerations related to peripheral timing support and DRAM structural constraints.

A. Scalability Analysis

The scalability of the proposed B2S framework arises directly from DRAM's hierarchical organization. Throughput and effective unary bitstream length expand along four orthogonal dimensions: column width, bank-level parallelism, operand precision, and optional intra-bank segmentation.

Column width dictates the number of unary bits generated per weighted comparison. Because a single `ACTIVATE` engages all bitline–SA pairs in parallel, increases in per-bank column count across DRAM generations translate directly into wider unary slices without altering array circuitry. Bank-level parallelism provides a second axis of throughput scaling. Each bank independently performs initialization, operand broadcast, and weighted comparison, allowing multiple conversions to proceed concurrently.

Operand precision scales linearly with the number of DUS rows. An n-bit operand requires n template rows and n weighted activations.

Intra-bank segmentation relaxes the baseline limitation that arises from RowClone's full-row granularity, which replicates a single operand bit across all columns and therefore yields only one unary bitstream per bank. To enable multiple independent streams, the template set is extended with segment-pattern rows that encode operand bits independently across predefined column segments. For example, partitioning a 1,024-bit row into four 256-bit segments requires sixteen rows that enumerate all 2^4 combinations of segment-level 0/1 patterns. During operand broadcast, each of the n operand bits is written using a single RowClone operation, but the source row is chosen from the segment-pattern set, allowing four segment-wise operand bits to be assigned in one step. The ADUS/SDUS templates and the time-weighted comparator remain unchanged, and the comparison consequently produces four independent 256-bit unary bitstreams. This extension increases the template footprint from 18 to 32 rows (0.055% to 0.10% of a 32,768-row bank). The same construction generalizes to finer segmentation—for example, eight 128-bit streams—by adding the corresponding pattern rows without modifying the DRAM array or comparator pipeline.

Overall, the framework scales naturally with existing DRAM structural boundaries. Because the 1T1C cell array, SAs, and decoders remain unmodified, all DUS and segmentation templates reside in standard DRAM rows. The required storage overhead is small and configurable (e.g., ~0.1% of rows per bank) and can be reclaimed when SC support is disabled, enabling scalable operation without altering the DRAM core.

Importantly, this storage overhead is fixed and does not grow with workload size. In GEMM, all left-hand operands use the ADUS template and all right-hand operands use the SDUS template, so each multiplication pairs streams from two different templates and satisfies pairwise decorrelation. Operands sharing a template (e.g., a_i and a_j) interact only through scaled addition, which is insensitive to input correlation [42]. One ADUS-SDUS pair therefore suffices for arbitrary GEMM dimensions. End-to-end inference on LeNet-5, ResNet-18, ResNet-50, VGG-16, and ViT-B/16 with a single fixed template pair ($N = 256$) achieves accuracy within 0.2% of FP32 across all five models, confirming that one template pair provides sufficient independence for practical large-scale workloads.

B. Comparative Analysis With External SNGs

For SC computation performed inside DRAM, both input operands must reside as bitstreams within the subarray before bitwise operations can proceed. To quantify this cost, we evaluate the end-to-end overhead of generating the 32,768-bit unary data required for one entry of a 16×16 GEMM tile. This operation uses 32 scalar operands—16 from a matrix row and 16 from the corresponding column—each converted into a 1,024-bit unary bitstream, yielding $32 \times 1{,}024 = 32{,}768$ bits (4 KB) that must reside in DRAM before in-memory AND operations can exploit full bitline parallelism. We decompose the total cost into: (i) bitstream generation (B2S), (ii) transfer into the DRAM compute region (4 KB), and (iii) in-DRAM unary multiplication via a single AAP. This decomposition ensures a fair comparison: external SNGs incur a DRAM write to place the bitstream in the compute region, whereas the in-DRAM SNG uses RowClone-based movement within the same bank.

Table IV summarizes the results. The in-DRAM SNG incurs a higher intrinsic generation cost (1.704 μJ) than the LFSR- and Sobol-based CMOS SNGs reported in prior work (0.003–0.005 μJ) [47], but avoids the dominant I/O transfer penalty: RowClone moves the 4 KB unary volume at only 0.040 μJ and 90 ns, whereas external SNGs must perform a full off-chip write, consuming 3.600 μJ and 1046 ns. The AND itself contributes only 0.0016 μJ and 80 ns. As a result, the total end-to-end cost of the in-DRAM SNG (1.7456 μJ, 4.4 μs) is more than 2× lower in both energy and latency than external LFSR/Sobol generators, whose transfer overhead dominates the pipeline.

Table IV also includes an SRAM-based SNG [48] and two representative memristive in-memory SNGs [22], [24], which achieve substantially lower cost by exploiting device-level programmability unavailable in commodity DRAM. Within the

TABLE IV
END-TO-END ENERGY/LATENCY PER TILE OUTPUT

		In-DRAM SNG	LFSR [47]	Sobol [47]	In-SRAM [48]	In-Mem. [22]	In-Mem. [24]
E (μJ)	Gen.	1.704	0.00311	0.00524	3.604	0.000287	0.000438
	Trans.	0.040	3.600	3.600	—	0	0
	Comp.	0.0016	0.0016	0.0016	—	Incl.	0.0000019
	Total	1.7456	3.60471	3.60684	>3.604	0.000287	0.00044
L (ns)	Gen.	4231	10160	20480	163840	4	10010
	Trans.	90	1046	1046	—	0	0
	Comp.	80	80	80	—	2	64
	Total	4401	11286	21606	>163840	6	10074

E: Energy, L: Latency.

DRAM-PIM context, where the relevant baseline is external CMOS SNGs whose bitstreams must traverse the memory channel, the proposed method reduces end-to-end energy and latency by more than 2×.

C. Trade-off Analysis With Binary In-DRAM PIM

We compare the proposed in-DRAM SNG framework with representative binary in-DRAM PIM primitives, using PIM-DRAM [49] and MIMDRAM [50] as baselines. A key difference lies in the utilization of bitline parallelism. In a bank with m bitline pairs, binary PIM executes each AAP/AP step across all m columns in parallel, advancing m independent n-bit multiplications through a multi-step schedule. In contrast, in-DRAM SC assigns N bitline pairs to represent an N-bit unary bitstream, so a single multiplication occupies N columns and completes in one AAP. Under a fixed array width of m columns, only m/N unary multiplications can proceed in parallel, reducing effective bitline parallelism from m to m/N.

To enable a fair comparison across these distinct parallelization models, all designs are normalized to the energy and latency of a single n-bit × n-bit multiplication. The AAP/AP complexity for the binary baselines follows their published models: $f_{\text{PIM-DRAM}}(n) = 3n^2 + 4(n-1)^3 + 4(n-1)$ for PIM-DRAM, and $f_{\text{MIMDRAM}}(n) = 11n^2 - 5n - 1$ for MIMDRAM. Because these counts correspond to computing m parallel multiplications across all bitline pairs in a bank, dividing by m yields the normalized per-multiply costs $F_{\text{PIM-DRAM}}(n)$ and $F_{\text{MIMDRAM}}(n)$. For in-DRAM SC, only m/N products are evaluated in parallel per AAP, giving a normalized computation cost of $F_{\text{SC-DRAM}}(n, N) = N/m$. Two unary lengths are evaluated: $L = 2^n$, which provides full n-bit precision, and $L = 2^{n-1}$, which halves the bitstream length at the cost of one bit of precision. Before accounting for generation and transfer costs, SC achieves lower per-multiply latency than binary PIM at $n \le 8$ due to its single-AAP execution; beyond this point the exponential growth of N causes SC cost to rise more rapidly than the polynomial scaling of binary PIM.

Fig. 11 presents the full end-to-end comparison, incorporating B2S generation, transfer, in-memory computation, and S2B conversion for all SC methods, thereby ensuring a fair comparison with binary PIM baselines that require neither B2S nor S2B. The S2B stage adopts the AGNI isolatency converter [34], which completes stochastic-to-binary conversion in a fixed 55 ns row-activation-to-latch cycle shared across $1{,}024/N$ parallel conversions per bank. At $n = 8$ ($N = 256$), the amortized per-conversion overhead is approximately

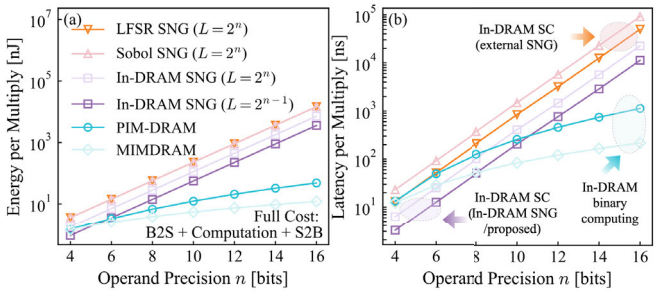


Fig. 11. End-to-end (a) energy and (b) latency per n-bit multiplication, including B2S generation, transfer, in-memory computation, and S2B conversion.

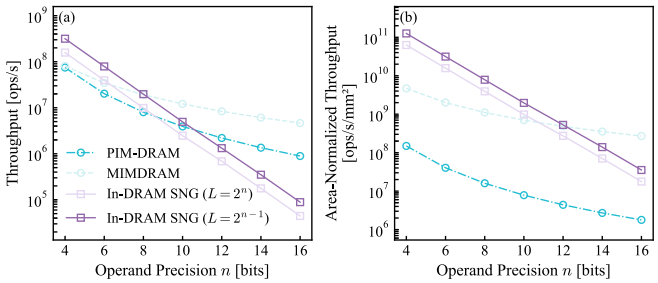


Fig. 12. (a) Throughput and (b) area-normalized throughput per n-bit multiplication, including B2S, computation, and S2B for SC pipelines.

13.75 ns and 0.36 pJ—negligible relative to the B2S generation and computation stages that dominate the pipeline.

In terms of energy (Fig. 11(a)), binary PIM achieves the lowest cost for most operand precisions, as it avoids unary bitstream formation entirely. The proposed in-DRAM SNG ranks second, benefiting from low-cost intra-bank RowClone movement, while external LFSR/Sobol SNGs incur significantly higher end-to-end costs due to the dominant off-chip write overhead.

In terms of latency (Fig. 11(b)), the proposed DRAM-resident SNG provides lower end-to-end latency than binary in-DRAM PIM approaches at low-to-moderate operand precisions ($n < 8$). At $n = 8$, the proposed in-DRAM SNG remains faster than PIM-DRAM and achieves latency on par with MIMDRAM. At higher operand precisions, the reduced m/N parallelism causes unary latency to grow more rapidly, allowing binary PIM to become the faster option. Including S2B does not shift these crossover points, as the fixed 55 ns conversion latency is negligible compared to the generation and computation costs that dominate at higher precisions.

For an honest performance comparison, we further evaluate area-normalized throughput, defined as the number of n-bit multiplications completed per second per unit silicon area beyond the unmodified DRAM bank.

The proposed in-DRAM SC pipeline incurs 731 μm^2 of B2S control logic plus 1,020 μm^2 of AGNI-based S2B circuitry, totaling ~1,751 μm^2 per bank; the off-chip B2S microsequencer (1,478 μm^2) is excluded as it resides outside the DRAM die and is shared across all banks. By comparison, PIM-DRAM and MIMDRAM require approximately 0.5 mm^2 and 0.0173 mm^2 of additional per-bank logic, respectively. Fig. 12 reports raw and area-normalized throughput across operand precisions. At $n = 8$, the half-length configuration achieves substantially higher area-normalized throughput than both PIM-DRAM and MIMDRAM. The advantage diminishes

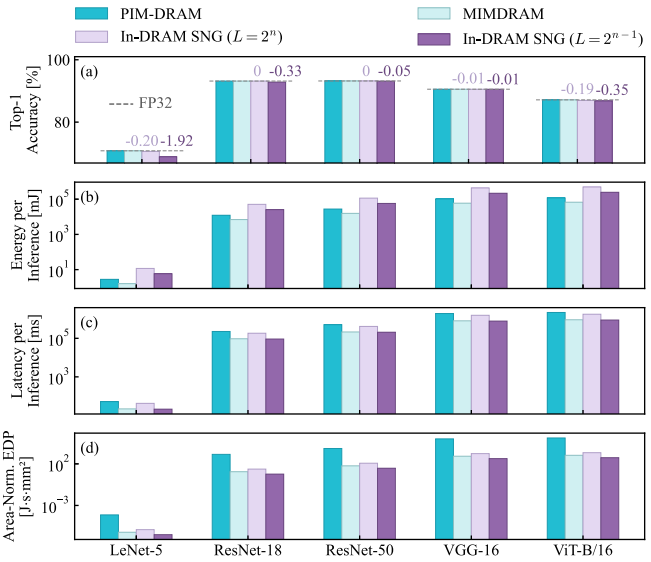


Fig. 13. Model-level evaluation on five benchmarks at n=8: (a) Top-1 accuracy, (b) energy per inference, (c) latency per inference, and (d) area-normalized EDP.

at higher precisions as unary length grows exponentially, mirroring the latency trends discussed above.

To extend the analysis from a single GEMM tile to realistic inference workloads, we evaluate five standard benchmarks spanning 0.42 M to 17.56 G MACs: LeNet-5, ResNet-18, ResNet-50, VGG-16, and ViT-B/16. Following the hybrid arithmetic strategy adopted by prior DRAM-PIM accelerators [51], [52], stochastic multiplication and first-level scaled addition use DUS bitstreams, while subsequent accumulation uses binary addition. All methods are evaluated at n=8.

The proposed in-DRAM SNG (L=2^n) achieves Top-1 accuracy within 0.2% of FP32 across all five models, as shown in Fig. 13(a). In terms of latency, the L=2^{n-1} configuration is comparable to MIMDRAM and approximately 2.4× faster than PIM-DRAM, as shown in Fig. 13(c). While per-inference energy is higher than binary PIM due to the inherent bitstream expansion (Fig. 13(b)), the area-normalized EDP reveals a complementary picture: because the in-DRAM SNG requires only ∼1,751 μm^2 of additional per-bank logic versus 0.5 mm^2 for PIM-DRAM, the proposed method at L=2^{n-1} achieves substantially lower AEDP across all five models (Fig. 13(d)), reflecting its minimal silicon footprint.

Overall, the comparison reveals a complementary set of trade-offs. External SNGs are fundamentally constrained by off-chip bitstream transfer overhead. In contrast, the proposed in-DRAM SNG offers lower latency and competitive energy at low-to-moderate precisions without requiring array-level modification, making it well suited for DRAM-PIM deployments. At higher precisions, binary in-DRAM PIM multipliers become more favorable due to their polynomial scaling. Together, these results indicate that deterministic in-DRAM unary bitstream generation provides a practical option for workloads that benefit from flexible precision and value low latency and DRAM-compatible execution.

D. Limitations and Practical Considerations

While the proposed deterministic in-DRAM SNG framework achieves competitive accuracy and balanced energy–latency behavior, several practical considerations arise from its reliance on DRAM structures and peripheral timing support. First, operand broadcast relies on RowClone-style full-row data movement, which precludes fine-grained operand placement. As a result, operand replication is restricted to template-based layouts within a subarray. Although intra-bank segmentation can partially alleviate this constraint by enabling multiple independent streams per bank, the required template footprint grows exponentially with the number of segments, limiting practical designs to moderate granularities (e.g., four or eight segments). Second, the time-weighted comparison mechanism requires per-bank timing support, including multi-phase wordline control and fine-grained delay tuning. Supporting such activation behavior may necessitate extensions to controller-visible command sequencing, increasing design and verification complexity relative to conventional DRAM interfaces.

Although corner and Monte Carlo simulations in Section III-C confirm robust sensing margins under process and temperature variation, scaled DRAM technologies and long-term manufacturing drift warrant further validation, as probabilistic systems are known to be sensitive to device-level non-idealities [53]. Additionally, recent experimental work demonstrates that dense multi-row activation patterns can degrade read-disturbance thresholds in commodity DRAM [54], motivating read-disturbance-aware activation scheduling and compute-region isolation in future deployments. Robust operation therefore benefits from per-bank timing calibration and conservative activation windows, with long-term characterization across DRAM generations remaining a topic for future investigation.

Overall, these considerations do not invalidate the proposed approach, but they delineate its practical operating envelope and motivate further study on robust timing control and scalable operand mapping within commodity DRAM constraints.

VI. Conclusion

This work presented a deterministic in-DRAM stochastic number generation framework that performs B2S conversion entirely within DRAM banks while leaving the 1T1C cell array, SAs, and row/column decoders unchanged. The design combines DRAM-resident DUS templates with a time-weighted comparison mechanism that encodes operand magnitude in wordline-on durations, enabling standard SAs to act as comparators. A three-phase per-bank B2S pipeline—comprising template initialization, RowClone-based operand and threshold broadcast, and time-weighted comparison—exploits intrinsic bitline and bank-level parallelism using peripheral timing support.

Bitstream-level evaluation shows that the proposed DUS sequences achieve SCC, ZCE, and unary arithmetic MAE comparable to Sobol across practical bitstream lengths. Operator- and task-level experiments on 3 × 3 Sobel filtering and a full Canny pipeline on BSDS500, together with inference on five CNN and ViT benchmarks, confirm that DUS preserves accuracy comparable to deterministic and LD References. System-level analysis indicates that performing unary bitstream generation inside DRAM eliminates the dominant bitstream-transfer overhead associated with external SNGs, reducing end-to-end energy and latency by more than 2× relative to external CMOS SNGs for representative GEMM tiles. For low-to-moderate operand precisions, the proposed DRAM-resident SNG also provides latency advantages over

binary in-DRAM PIM approaches whose operation counts scale superlinearly with precision. Collectively, these results indicate that deterministic in-DRAM unary bitstream generation offers a viable and DRAM-compatible path for integrating SC into PIM architectures.

REFERENCES

[1] S. C. Smithson, N. Onizawa, B. H. Meyer, W. J. Gross, and T. Hanyu, "Efficient CMOS invertible logic using stochastic computing," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 66, no. 6, pp. 2263–2274, Jun. 2019.

[2] Z. Chen, Y. Ma, and Z. Wang, "Hybrid stochastic-binary computing for low-latency and high-precision inference of CNNs," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 69, no. 7, pp. 2707–2720, Jul. 2022.

[3] P. Wang et al., "MS-SCIM: A mixed-signal stochastic computing-in-memory paradigm for information security," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 72, no. 7, pp. 3226–3235, Jul. 2025.

[4] Z. Lin, G. Xie, W. Xu, J. Han, and Y. Zhang, "Accelerating stochastic computing using deterministic halton sequences," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 68, no. 10, pp. 3351–3355, Oct. 2021.

[5] Y. Kiran and M. Riedel, "A scalable, deterministic approach to stochastic computing," in *Proc. Great Lakes Symp. VLSI*, Irvine, CA, USA, Jun. 2022, pp. 45–51.

[6] H. Nair et al., "TuGEMM: Area-power-efficient temporal unary GEMM architecture for low-precision edge AI," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, Monterey, CA, USA, May 2023, pp. 1–5.

[7] E. Becle, G. Prenat, P. Talatchian, L. Anghel, and I.-L. Prejbeanu, "A fast, energy efficient and tunable magnetic tunnel junction based bitstream generator for stochastic computing," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 69, no. 8, pp. 3251–3259, Aug. 2022.

[8] M. R. Alam, M. H. Najafi, N. Taherinejad, M. Imani, and R. Gottumukkala, "Stochastic computing in beyond Von-Neumann era: Processing bit-streams in memristive memory," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 69, no. 5, pp. 2423–2427, May 2022.

[9] V. Seshadri et al., "RowClone: Fast and energy-efficient in-DRAM bulk data copy and initialization," in *Proc. 46th Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Davis, CA, USA, Dec. 2013, pp. 185–197.

[10] V. Seshadri et al., "Ambit: In-memory accelerator for bulk bitwise operations using commodity DRAM technology," in *Proc. 50th Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Cambridge, MA, USA, Oct. 2017, pp. 273–287.

[11] C. F. Frasser, M. Roca, and J. L. Rosselló, "Optimal stochastic computing randomization," *Electronics*, vol. 10, no. 23, p. 2985, Dec. 2021.

[12] V. Sehwag, N. Prasad, and I. Chakrabarti, "A parallel stochastic number generator with bit permutation networks," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 65, no. 2, pp. 231–235, Feb. 2018.

[13] S. Hu, K. Han, F. Wang, and J. Hu, "Hybrid stochastic LDPC decoder with fully correlated stochastic computation," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 69, no. 9, pp. 3643–3654, Sep. 2022.

[14] M. S. Moghadam, S. Aygun, M. R. Alam, and M. H. Najafi, "P2LSG: Powers-of-2 low-discrepancy sequence generator for stochastic computing," in *Proc. 29th Asia South Pacific Design Autom. Conf. (ASP-DAC)*, Jan. 2024, pp. 38–45.

[15] D. Jenson and M. Riedel, "A deterministic approach to stochastic computation," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, Austin, TX, USA, Nov. 2016, pp. 1–8.

[16] P. Schober, M. H. Najafi, and N. TaheriNejad, "High-accuracy multiply-accumulate (MAC) technique for unary stochastic computing," *IEEE Trans. Comput.*, vol. 71, no. 6, pp. 1425–1439, Jun. 2022.

[17] S. Yu and S. X.-D. Tan, "Scaled-CBSC: Scaled counting-based stochastic computing multiplication for improved accuracy," in *Proc. 59th ACM/IEEE Design Autom. Conf.*, San Francisco, CA, USA, Jul. 2022, pp. 1003–1008.

[18] D. Wu, J. Li, R. Yin, H. Hsiao, Y. Kim, and J. S. Miguel, "UGEMM: Unary computing for GEMM applications," *IEEE Micro*, vol. 41, no. 3, pp. 50–56, May 2021.

[19] P. Vellaisamy et al., "TubGEMM: Energy-efficient and sparsity-effective temporal-unary-binary based matrix multiply unit," in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI (ISVLSI)*, Sep. 2023, pp. 1–6.

[20] M. Tasnim, S. Sachdeva, Y. Liu, and S. X. D. Tan, "Hybrid temporal computing for lower power hardware accelerators," in *Proc. 30th Asia South Pacific Design Autom. Conf.*, Japan, Jan. 2025, pp. 237–244.

[21] C. Lammie, J. K. Eshraghian, W. D. Lu, and M. R. Azghadi, "Memristive stochastic computing for deep learning parameter optimization," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 68, no. 5, pp. 1650–1654, May 2021.

[22] M. R. Alam, M. H. Najafi, and N. TaheriNejad, "Exact stochastic computing multiplication in memristive memory," *IEEE Design Test*, vol. 38, no. 6, pp. 36–43, Dec. 2021.

[23] S. Gupta et al., "SCRIMP: A general stochastic computing architecture using ReRAM in-memory processing," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Grenoble, France, Mar. 2020, pp. 1598–1601.

[24] J. P. C. de Lima, M. S. Moghadam, S. Aygun, J. Castrillon, M. H. Najafi, and A. A. Khan, "All-in-memory stochastic computing using ReRAM," in *Proc. 62nd ACM/IEEE Design Autom. Conf. (DAC)*, San Francisco, CA, USA, Jun. 2025, pp. 1–7.

[25] F. Gao, G. Tziantzioulis, and D. Wentzlaff, "ComputeDRAM: In-memory compute using off-the-shelf DRAMs," in *Proc. 52nd Annu. IEEE/ACM Int. Symp. Microarchitecture*, Columbus, OH, USA, Oct. 2019, pp. 100–113.

[26] İ. E. Yüksel et al., "Simultaneous many-row activation in off-the-shelf DRAM chips: Experimental characterization and analysis," in *Proc. 54th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw. (DSN)*, Brisbane, Australia, Jun. 2024, pp. 99–114.

[27] İ. E. Yüksel et al., "Functionally-complete Boolean logic in real DRAM chips: Experimental characterization and analysis," in *Proc. IEEE Int. Symp. High-Perform. Comput. Archit. (HPCA)*, Mar. 2024, pp. 280–296.

[28] A. Olgun et al., "In-DRAM true random number generation using simultaneous multiple-row activation: An experimental study of real DRAM chips," in *Proc. 43rd IEEE Int. Conf. Comput. Des. (ICCD)*, Oct. 2025, pp. 754–763.

[29] K. Humood, B. Mohammad, and H. Abunahla, "DTRNG: Low cost and robust true random number generator using DRAM weak write scheme," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2021, pp. 1–5.

[30] H. Niederreiter, "Low-discrepancy and low-dispersion sequences," *J. Number Theory*, vol. 30, no. 1, pp. 51–70, Sep. 1988.

[31] W. J. Morokoff and R. E. Caflisch, "Quasi-random sequences and their discrepancies," *SIAM J. Sci. Comput.*, vol. 15, no. 6, pp. 1251–1279, 1994.

[32] V. Sinescu and S. Joe, "Good lattice rules based on the general weighted star discrepancy," *Math. Comput.*, vol. 76, no. 258, pp. 989–1004, Dec. 2006.

[33] S. Angizi and D. Fan, "ReDRAM: A reconfigurable processing-in-DRAM platform for accelerating bulk bit-wise operations," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, Westminster, CO, USA, Nov. 2019, pp. 1–8.

[34] S. M. Shivanandamurthy, S. S. Vatsavai, I. Thakkar, and S. A. Salehi, "AGNI: In-situ, iso-latency stochastic-to-binary number conversion for in-DRAM deep learning," in *Proc. 24th Int. Symp. Quality Electron. Design (ISQED)*, Santa Clara, CA, USA, Apr. 2023, pp. 1–8.

[35] H. Shin, R. Park, and J. W. Lee, "A processing-using-memory architecture for commodity DRAM devices with enhanced compatibility and reliability," in *Proc. 43rd IEEE/ACM Int. Conf. Computer-Aided Design*, Jersey, Oct. 2024, pp. 1–10.

[36] P. Huang et al., "Offset-compensation high-performance sense amplifier for low-voltage DRAM based on current mirror and switching point," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 69, no. 4, pp. 2011–2015, Apr. 2022.

[37] M. H. Najafi and D. J. Lilja, "High quality down-sampling for deterministic approaches to stochastic computing," *IEEE Trans. Emerg. Topics Comput.*, vol. 9, no. 1, pp. 7–14, Jan. 2021.

[38] Y. Kim, W. Yang, and O. Mutlu, "Ramulator: A fast and extensible DRAM simulator," *IEEE Comput. Archit. Lett.*, vol. 15, no. 1, pp. 45–49, Jan. 2016.

[39] K. Chandrasekar, B. Akesson, and K. Goossens, "Improved power modeling of DDR SDRAMs," in *Proc. 14th Euromicro Conf. Digit. Syst. Design*, Aug. 2011, pp. 99–108.

[40] S. Liu and J. Han, "Energy efficient stochastic computing with Sobol sequences," in *Proc. Design, Autom. Test Europe Conf. Exhib. (DATE)*, 2017, pp. 650–653.

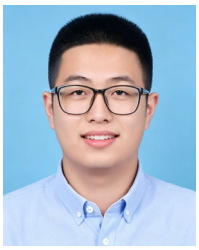
[41] M. S. Moghadam, S. Aygun, M. R. Alam, J. I. Schmidt, M. H. Najafi, and N. Taherinejad, "Accurate and energy-efficient stochastic computing with Van der Corput sequences," in *Proc. 18th ACM Int. Symp. Nanosc. Archit.*, Dec. 2023, pp. 1–6.

[42] A. Alaghi and J. P. Hayes, "Exploiting correlation in stochastic circuit design," in *Proc. IEEE 31st Int. Conf. Comput. Design (ICCD)*, Oct. 2013, pp. 39–46.

[43] H. Hsiao, J. S. Miguel, Y. Hara-Azumi, and J. Anderson, "Zero correlation error: A metric for finite-length bitstream independence in stochastic computing," in *Proc. 26th Asia South Pacific Design Autom. Conf. (ASP-DAC)*, Tokyo, Japan, Jan. 2021, pp. 260–265.
[44] H. Joe and Y. Kim, "Novel stochastic computing for energy-efficient image processors," *Electronics*, vol. 8, no. 6, p. 720, Jun. 2019.
[45] J. Canny, "A computational approach to edge detection," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. PAMI-8, no. 6, pp. 679–698, Nov. 1986.
[46] P. Arbeláez, M. Maire, C. Fowlkes, and J. Malik, "Contour detection and hierarchical image segmentation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 33, no. 5, pp. 898–916, May 2011.
[47] S. Liu and J. Han, "Toward energy-efficient stochastic circuits using parallel Sobol sequences," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 26, no. 7, pp. 1326–1339, Jul. 2018.
[48] H. Shi, Z. Yu, T. Zhang, J. Han, Y. Yang, and S. Liu, "An SRAM-based stochastic number generator for stochastic computing," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, Nov. 2025, pp. 1–5.
[49] S. Roy, M. Ali, and A. Raghunathan, "PIM-DRAM: Accelerating machine learning workloads using processing in commodity DRAM," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 11, no. 4, pp. 701–710, Dec. 2021.
[50] 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 *Proc. IEEE Int. Symp. High-Perform. Comput. Archit. (HPCA)*, Mar. 2024, pp. 186–203.
[51] S. Li et al., "SCOPE: A stochastic computing engine for DRAM-based in-situ accelerator," in *Proc. 51st Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Fukuoka, Japan, Oct. 2018, pp. 696–709.
[52] S. M. Shivanandamurthy, I. G. Thakkar, and S. A. Salehi, "ATRIA: A bit-parallel stochastic arithmetic based accelerator for in-DRAM CNN processing," in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI (ISVLSI)*, Tampa, FL, USA, Jul. 2021, pp. 200–205.
[53] A. Haroon, R. K. Ghosh, and S. Saurabh, "Impact of non-idealities on the behavior of probabilistic computing: Theoretical investigation and analysis," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 71, no. 12, pp. 6279–6291, Dec. 2024.
[54] I. E. Yuksel et al., "PuDHammer: Experimental analysis of read disturbance effects of processing-using-DRAM in real DRAM chips," in *Proc. 52nd Annu. Int. Symp. Comput. Archit.*, Tokyo, Japan, Jun. 2025, pp. 757–775.

Shiyu Xia received the M.Eng. degree from the University of Chinese Academy of Sciences, Beijing, China, in 2021. She is currently pursuing the Ph.D. degree with Shanghai Jiao Tong University, Shanghai, China. From 2023 to 2025, she was an Intern with Changxin Memory Technologies (CXMT), Hefei, China. Her research interests include processing-in-memory, stochastic computing, DRAM architecture, and memory system design.

Chen Zhang received the Ph.D. degree in EECS from Peking University, Beijing, China, in 2017. He is currently an Associate Professor with Shanghai Jiao Tong University. He served as a Senior Researcher with Microsoft Research Asia and a GPGPU Architect with Alibaba. His research interests include computer architectures and heterogeneous computing for cloud and edge AI systems. He has published about 40 papers in ISCA/MICRO/HPCA/FPGA/DAC/ICCAD. He has received two Ten-Year Retrospective Most Influential Paper Awards from FPGA'2015 and ICCAD'2016 and three best paper awards from TCAD'2019, ISCA'2025, and ISEDA'2025.



Mingzhao Yang received the M.S. degree in electronic information from Tongji University, Shanghai, China, in 2026. Since 2014, he has been with Semiconductor Manufacturing International Corporation (SMIC), Shanghai GTA Semiconductor, and ChangXin Memory Technologies (CXMT), where he has been involved in WAT/CP testing, ESD circuit design, and heterogeneous integration chip design. He is currently a Principal Engineer. His research interests include heterogeneous integration chip design, machine learning for wafer electrical map prediction, and test circuit development for 3-D chiplet integration.

Hao Meng received the B.Sc. degree in physics from Jilin University, Changchun, China, in 1998, and the Ph.D. degree in electrical engineering from the University of Minnesota, Minneapolis, MN, USA, in 2007. He is currently the Research and Development Director with ChangXin Memory Technologies (CXMT). He has authored or co-authored more than 70 scientific articles and filed over 80 patents. His research interests include memory technology.

Mingyuan Liu (Member, IEEE) received the Ph.D. degree in materials science from Shanghai Jiao Tong University, Shanghai, China, in 2004. He is currently the Vice President and the Head of the Technology Center of Excellence, ChangXin Memory Technologies (CXMT). With 22 years of experience in the research and development of integrated process technology, he mainly focuses on technology development of advanced Logic, DRAM, and NAND. Having worked in memory technology research and development for years, he has published nearly 20 technical articles and obtained over 40 patents in both China and USA.

Zhigang Ji (Member, IEEE) received the B.Eng. degree in electrical engineering from Tsinghua University, Beijing, China, in 2003, the M.Eng. degree in microelectronics from Peking University, Beijing, in 2006, and the Ph.D. degree in microelectronics from Liverpool John Moores University, Liverpool, U.K., in 2010. He was a Reader with Liverpool John Moores University. He is currently a Professor with the School of Electronic Information and Electrical Engineering, Shanghai Jiao Tong University, Shanghai, China. He has authored or co-authored over 100 journal articles. His research interests include micro/nano electronic devices, reliability physics, circuit/device co-design for reliability, novel paradigm computing, and hardware security.