SCoRe: Stochastic Compressed Representations

Paridhi Gupta, Zhewen Pan, Julie Hsiao, Joshua San Miguel
University of Wisconsin–Madison
pgupta58@wisc.edu, zhewen.pan@wisc.edu, julie.hsiao@wisc.edu, jsanmiguel@wisc.edu

***Abstract*—Stochastic Computing (SC) offers significant advantages in hardware efficiency, however, data storage remains a critical bottleneck, as buffering intermediate results in their native format is prohibitively expensive. While compressing stochastic streams into binary values alleviates storage constraints, it discards vital statistical information and temporal properties like independence and Streaming Accuracy, which require expensive hardware to regenerate. To bridge the trade-off between storage density and information fidelity, this paper evaluates different compression schemes based on their ability to preserve accuracy and independence, as well as their hardware complexity.**

I. Introduction

Stochastic Computing (SC) has emerged as a powerful alternative to traditional binary logic, particularly for low precision applications. By representing values as probabilistic bitstreams, SC enables complex arithmetic operations to be performed using simple logic gates. This paradigm offers significant advantages in terms of hardware efficiency, drastically reducing both silicon area and power consumption compared to conventional binary implementations.

However, data storage remains a critical bottleneck. Many SC architectures, particularly in deep neural networks and iterative processing, require the buffering of intermediate results between computation stages [5], [8], [9]. Storing these values in their native stochastic format is prohibitively expensive in terms of memory area. Conversely, converting these streams to binary for compact storage is energy-intensive, consuming significant power that undermines the inherent efficiency gains of the SC paradigm.

Furthermore, compressing stochastic streams into binary values results in the loss of vital statistical information. Binary representation captures only the scalar probability of the stream, discarding temporal properties such as bitstream Independence and Streaming Accuracy. These properties are often deliberately engineered or naturally evolved during computation to ensure stability and correctness. Consequently, regeneration from binary is an approximate process and recovering the original correlation or stability characteristics requires expensive additional hardware, such as synchronizers and de-correlators.

To address these challenges and bridge the gap illustrated in Figure 1 this work explores the design space of stochastic compression to navigate the conflict between storage efficiency, information fidelity and hardware complexity. Through characterization of various encoding schemes, this study clarifies the suitability of each approach for different accuracy and hardware constraints.

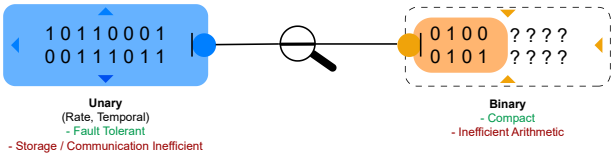


Fig. 1: Stochastic compression design space.

II. Background

In stochastic computing, the information carried by a bitstream is defined by its probability value. However, the specific arrangement of bits within the stream significantly impacts computational accuracy and latency. This section reviews key metrics used to characterize these properties.

A. Streaming Accuracy

Streaming Accuracy [6] measures the early terminability or stability of a stochastic bitstream. This metric is useful because it indicates how accurately a partial bitstream reflects the target value at any arbitrary point during processing, rather than requiring the entire sequence to be completed. Maintaining high streaming accuracy is useful as it allows for flexible latency as computations can be terminated early to save energy or time while still retaining a valid approximation of the result.

B. Independence

Beyond the accuracy of individual streams, the statistical relationship between pairs of bitstreams is critical for computation. Stochastic logic gates rely on specific input dependencies to function correctly. For example, an AND gate performs accurate multiplication only when its input sequences are uncorrelated/independent. If the stored data loses these structural properties during compression, the retrieved streams may yield significant computational errors or require expensive hardware to regenerate the necessary independence or correlation. Therefore, preserving these relationships ensures that decompressed data is immediately usable by downstream logic. Two metrics are used to characterize this:

1) Stochastic Cross Correlation (SCC): SCC [1] quantifies the specific degree of correlation between two bitstreams by normalizing their covariance.

2) Zero Correlation Error (ZCE): ZCE [7] specifically evaluates the independence of bitstreams, designed to distinguish true structural independence from the quantization noise inherent in finite-length sequences.

III. Compression and Decompression

Building on the definitions provided in Section II, this section details the implementation of five distinct compression architectures that we explore and characterize in this work.

A. Binary

The conventional baseline for compressing a stochastic bitstream is to convert it into a fixed-point binary integer representing the total count of ones (popcount). This method achieves the highest compression density but discards all temporal and spatial information, effectively erasing the stream's correlation and stability characteristics. We evaluate two distinct strategies for regenerating the stochastic stream from this binary value:

1) Random: The bitstream is regenerated using a standard Stochastic Number Generator (SNG), typically implemented with a Linear Feedback Shift Register (LFSR) [2]. This produces a pseudo-random sequence that matches the target probability but possesses no guaranteed correlation or stability properties, effectively resetting the stream's statistical history.

2) Streaming Accurate: The bitstream is reconstructed using a deterministic generator designed to maximize streaming accuracy [6].

B. Dictionary

In this scheme, the input unary bitstream is segmented into fixed-size chunks of length L. We have a pre-defined codebook which is a static lookup table composed of L-bit binary sequences, referred to as keywords. To compress the number we store the index of the keyword that best matches each chunk. To guarantee complete value coverage, the codebook is populated with at least one keyword for every possible Hamming weight (from 0 to L). This ensures that any probability value resolvable within the window size can be represented. To decompress we simply retrieve and concatenate these keywords to reconstruct the stream. We explore two distinct strategies for selecting the optimal keyword for a given input chunk.

1) Value Based: This strategy prioritizes the numerical fidelity of the stream. An input chunk is mapped to the keyword that minimizes the difference in value. In cases where multiple keywords offer the same value proximity, the selection is refined by choosing the keyword with the smallest Hamming distance to the input chunk.

2) Hamming: This strategy prioritizes the structural integrity of the bitstream, which is critical for preserving correlation properties like SCC. An input chunk is directly mapped to the keyword that minimizes the Hamming distance.

A key advantage of the dictionary scheme is that the hardware complexity for the compression and decompression logic remains constant and does not scale with the bitstream length. Notably, if the Value Based compression strategy is chosen, the system exhibits no value error upon decompression and should maintain a significant degree of the original correlation and streaming accuracy.

C. Contingency Tables

This scheme utilizes the insights from [3], which proposes representing stochastic numbers through contingency tables (CTs) to save both computation time and memory. Rather than storing individual bit sequences, this method compresses two bitstreams jointly by explicitly storing their correlation statistics. For a pair of input streams, the system counts the four possible bit-pair combinations—$(1,1)$, $(1,0)$, $(0,1)$, and $(0,0)$—which are stored as the scalar variables a, b, c, and d. These four counts form the contingency table that serves as the compressed representation.

To use CT, we compress the data using the counts above and decompress it by generating a random sequence of bit-pairs that matches the stored CT values. Consequently, this method retains perfect value accuracy and allows for the preservation or adjustment of correlation between the streams without the need for bit-by-bit processing.

D. Lower Precision

This is the most direct method of compression, achieved by simply truncating the stochastic bitstream to a shorter length L_{short}. While this reduces the storage requirement, it inherently lowers the resolution of the represented value. To decompress, the truncated stream is restored to the original length L by repeating the sequence L/L_{short} times. Due to these straightforward operations, the compression and decompression hardware is very simple.

E. Odd Bits

This scheme compresses the stream by spatially subsampling the data. We retain only the bits located at odd indices, discarding the even-indexed bits to halve the storage requirement. To mitigate the information loss, a single flag bit is generated to summarize the discarded data; this flag is set to 1 if the majority of the even bits were 1, and 0 otherwise.

During decompression, the odd bits are restored to their original positions. The even positions are then populated by the stored flag bit, providing a crude but low-cost approximation of the missing data.

IV. Characterization and Evaluation

A. Methodology

We evaluate the compression and decompression schemes based on their ability to preserve accuracy and independence (Section IV-B), as well as their underlying hardware complexity (Section IV-D). For these evaluations, the bitstream length is fixed at 256. The dictionary-based schemes utilize a chunk size of 4 and a codebook size of 8, while the lower-precision scheme is configured to store 32 bits unless otherwise specified.

B. Accuracy and Correlation

1) Streaming Accuracy Loss: In these experiments, we use streaming accurate bitstreams generated by the SA bitstream generator [6] as inputs to the compressor. We quantify the discrepancy between input and output as SA loss and value loss. Figure 2b and Figure 2a illustrate these losses relative to

the bitstream value (P), while Table I summarizes the average SA and value loss for each evaluated scheme.

All binary-based designs, namely Binary_SA, Binary_Random, and CT exhibit low or zero value loss. These schemes utilize accumulators as lossless compressors and bitstream generators as decompressors, a combination that generally ensures high accuracy (though Binary_SA does incur a negligible value loss of 0.002). Notably, Binary_SA maintains perfect SA. Conversely, Binary_Random suffers an SA loss of 0.12, illustrating the inherent SA gap between LFSR-generated bitstreams and SA-perfect ones.

Sampling-based schemes incur significantly higher value loss than their binary counterparts. In the Odd_Bits scheme, even bits are interpolated using a holistic bias bit. While this interpolation typically perturbs the bitstream value by up to 0.25 near $P = 0.5$, the alternating 0s and 1s of streaming accurate bitstreams at that value ensure the odd bits and bias bit are inverses, leading to perfect interpolation. Meanwhile, the Lower_Precision scheme uses an n-bit partial bitstream to represent the entire sequence. Due to the nature of streaming accurate inputs, this partial bitstream can only represent $n+1$ values without loss, which explains the 32 periodic triangular ripples observed in Figure 2b when $n = 32$.

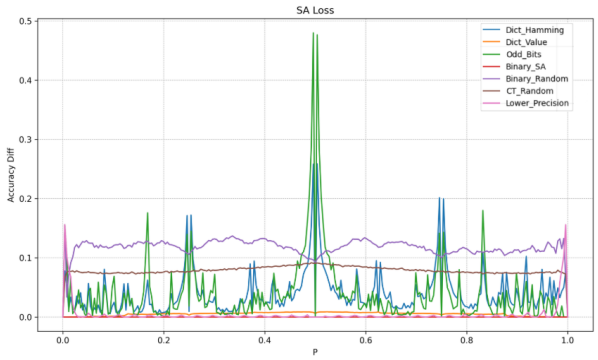
Dict. Val. demonstrates low value loss because its codebook keywords are selected to prioritize value preservation for each chunk. Because each chunk maintains its value during processing, Dict. Val. also preserves SA at the chunk granularity. In contrast, Dict. Ham. prioritizes maintaining a low Hamming distance from the original bitstream, which results in greater value perturbation.

Method	Val Diff	Streaming Acc Diff
Dict. Val.	0.0000	0.0050
Binary SA	0.0000	0.0000
CT Random	0.0000	0.0788
Binary Random	0.0020	0.1175
Lower Precision	0.0078	0.0045
Dict. Ham.	0.0694	0.0403
Odd Bits	0.1240	0.0404

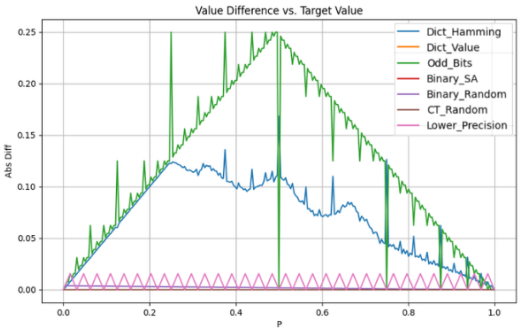
TABLE I: Average SA and value loss.

2) Independence Loss: In these experiments, we use decorrelated bitstream pairs as input and measure the Zero-Correlation Error (ZCE) for both input and output pairs, defining independence loss as the difference between them. Figure 3 displays the input and output ZCE for all schemes, excluding the CT scheme, which is designed to perfectly preserve correlation, while Table II provides a comprehensive summary.

The input ZCE (x-axis) shows a narrow spread due to the effectiveness of the decorrelation preprocessing, while the relative spread of the output ZCE (y-axis) denotes the independence loss. Empirically, both dictionary schemes are superior at preserving independence for decorrelated bitstream pairs, with the Dict. Val. scheme performing particularly well. Consequently, the numerical loss of AND multiplication, which relies on input independence, also remains low for Dict. Val..



(a) SA loss.

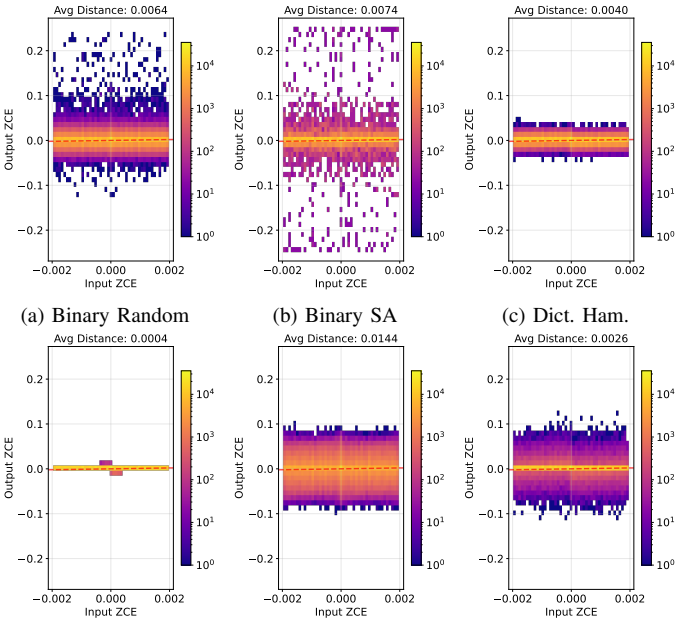


(b) Value loss

Fig. 2: SA and value loss distribution per bitstream value (P).

Metric	Bin.Rand.	Bin.SA	Dict.Ham.	Dict.Val.	Samp.	Lo.Pr.
ZCE	0.0064	0.0074	0.0040	0.0004	0.0144	0.0026
AND	0.0082	0.0090	0.0226	0.0008	0.1086	0.2242

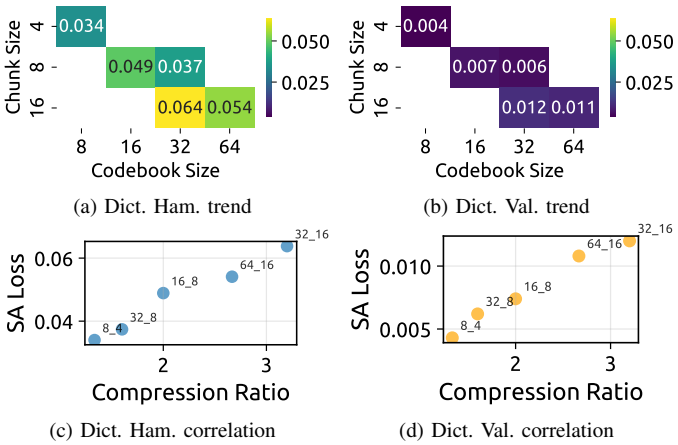
TABLE II: ZCE and AND multiplication numerical loss.



(a) Binary Random (b) Binary SA (c) Dict. Ham.

(d) Dict. Val. (e) Odd Bits (f) Lower Precision

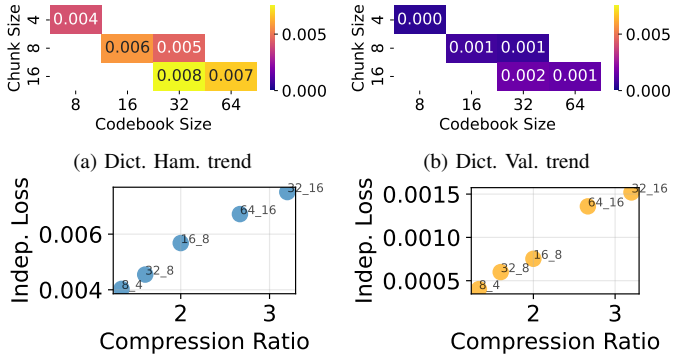
Fig. 3: Output and input ZCE distribution. Color scale represents density of points.



(a) Dict. Ham. trend (b) Dict. Val. trend

(c) Dict. Ham. correlation (d) Dict. Val. correlation

Fig. 4: SA loss sensitivity w.r.t. codebook size and chunk size.



(a) Dict. Ham. trend (b) Dict. Val. trend

(c) Dict. Ham. correlation (d) Dict. Val. correlation

Fig. 5: Independence loss sensitivity w.r.t. codebook size and chunk size.

C. Sensitivity Study

We next examine the impact of more aggressive compression, achieved via larger codebooks and chunk sizes, on SA and independence loss. As in the previous section, the SA results (Figure 4) are based on SA input bitstreams, while the independence results (Figure 5) utilize decorrelated input bitstreams.

With more aggressive compression, achieved by decreasing the number of codebook entries and/or increasing the chunk size, the reconstructed bitstream exhibits a greater loss of the original bitstream's properties. This trend is evident in both SA (Figure 4a and 4b) and ZCE (Figure 5a and 5b) for the Dict. Val. and Dict. Ham. schemes. However, Dict. Val. consistently shows lower loss. As demonstrated in Figure 4c, 4d, 5c, and 5d, both SA and independence loss correlate strongly with the compression ratio.

D. Hardware

We implement two variants (4×8 and 8×16) of dictionary scheme and CT scheme in RTL and synthesize them using Synopsys design compiler under 45nm technology. The area (μm^2) and power (μW) results for a bitstream length (L) of 256, broken down into compressor (comp.) and decompressor (decomp.) are shown in table III. We assume that compression and decompression each take L cycles. We include serializer and deserializer hardware in dictionary schemes for fairness. Our binary baseline uses an accumulator as compressor and a LFSR as decompressor. The energy results are proportional to power thus omitted here.

TABLE III: Area and power results assuming L=256 bitstreams.

Scheme	Area (μm^2)			Power (μW)		
	Comp.	Decomp.	Total	Comp.	Decomp.	Total
Binary	269.85	650.45	920.30	0.23	0.62	0.85
Dict. 4×8	239.34	218.22	457.57	0.24	0.28	0.51
Dict. 8×16	467.89	465.08	932.97	0.40	0.61	1.02
CT	460.15	925.46	1385.61	0.46	0.83	1.29

The compressor and decompressor are essentially hardwired combinational encoding and decoding logic. As shown in Table III, Dict. 4×8's compressor has similar area and power to that of binary baseline. However, the decompressor complexity is significantly lower. Together, it takes only 50% area and 60% of the binary baseline. Though previous section shows that CT scheme preserves perfect correlation, it takes 1.5× area than the binary baseline. Note that the CT scheme handles a pair of bitstreams so we halve its cost for fair comparison.

V. Related Work

Cambicron-U [5] introduces a new compression format in Cambricon-U by storing numbers as skew-binary. This representation reduces accumulation power because only 2–3 bits per cycle are activated during accumulation, unlike binary counters which can activate all bits. The values are eventually converted to binary for storage.

Efficient bitstream generation. Numerous works have focused on optimizing Stochastic Number Generators (SNGs) to reduce hardware complexity without sacrificing value accuracy. For instance, [4], [10] propose minimum probability conversion circuits (PCCs) and shared random number sources (RNS) that significantly lower the transistor count compared to traditional comparator-based designs. uBrain [11] proposes low cost temporal bitstream generator directly from analog signal.

VI. Conclusion

This work presents a characterization study of stochastic bitstream compression schemes, with a focus on tradeoffs between storage efficiency, information fidelity, and hardware complexity. Through evaluation of streaming accuracy, value, and independence loss across the input and output bitstream, we show that different schemes preserve bitstream properties at various levels. Binary as a compression scheme achieves the highest storage efficiency but comes at the cost of losing correlation and independence, requiring additional property manipulation hardware. Sampling-based schemes, though simple, incur more value and independence loss. Dictionary-based schemes preserve a significant degree of value and independence at a moderate hardware cost. These results show that stochastic compression involves fundamental tradeoffs between information, complexity, and efficiency.

REFERENCES

[1] A. Alaghi and J. P. Hayes, "Exploiting correlation in stochastic circuit design," in *2013 IEEE 31st International Conference on Computer Design (ICCD)*, 2013, pp. 39–46.

[2] ——, "Survey of stochastic computing," *ACM Trans. Embed. Comput. Syst.*, vol. 12, no. 2s, May 2013. [Online]. Available: https://doi.org/10.1145/2465787.2465794

[3] S. Aygun and E. O. Gunes, "Utilization of contingency tables in stochastic computing," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 69, no. 6, pp. 2942–2946, 2022.

[4] C. Collinsworth and S. A. Salehi, "Stochastic number generators with minimum probability conversion circuits," in *2021 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2021, pp. 49–54.

[5] H. Guo, Y. Zhao, Z. Li, Y. Hao, C. Liu, X. Song, X. Li, Z. Du, R. Zhang, Q. Guo, T. Chen, and Z. Xu, "Cambricon-u: A systolic random increment memory architecture for unary computing," in *2023 56th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2023, pp. 424–437.

[6] H. Hsiao, J. S. Miguel, and J. Anderson, "Streaming accuracy: Characterizing early termination in stochastic computing," in *2022 27th Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2022, pp. 320–325.

[7] 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 *2021 26th Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2021, pp. 260–265.

[8] Y. Liu, L. Liu, F. Lombardi, and J. Han, "An energy-efficient and noise-tolerant recurrent neural network using stochastic computing," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 9, pp. 2213–2221, 2019.

[9] G. Maor, X. Zeng, Z. Wang, and Y. Hu, "An fpga implementation of stochastic computing-based lstm," in *2019 IEEE 37th International Conference on Computer Design (ICCD)*, 2019, pp. 38–46.

[10] S. A. Salehi, "Low-cost stochastic number generators for stochastic computing," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, no. 4, pp. 992–1001, 2020.

[11] D. Wu, J. Li, Z. Pan, Y. Kim, and J. S. Miguel, "ubrain: a unary brain computer interface," in *Proceedings of the 49th Annual International Symposium on Computer Architecture*, ser. ISCA '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 468–481. [Online]. Available: https://doi.org/10.1145/3470496.3527401