Sobol Sequence Optimization for Hardware-Efficient Vector Symbolic Architectures

Sercan Aygun, *Member, IEEE* and M. Hassan Najafi *Senior Member, IEEE*

***Abstract*—Hyperdimensional computing (HDC) is an emerging computing paradigm with significant promise for efficient and robust learning. In HDC, objects are encoded with high-dimensional vector symbolic sequences called hypervectors. The quality of hypervectors, defined by their distribution and independence, directly impacts the performance of HDC systems. Despite a large body of work on the processing parts of HDC systems, little to no attention has been paid to data encoding and the quality of hypervectors. Most prior studies have generated hypervectors using inherent random functions, such as MATLAB's or Python's random function. This work introduces an optimization technique for generating hypervectors by employing quasi-random sequences. These sequences have recently demonstrated their effectiveness in achieving accurate and low-discrepancy data encoding in stochastic computing systems. The study outlines the optimization steps for utilizing Sobol sequences to produce high-quality hypervectors in HDC systems. An optimization algorithm is proposed to select the most suitable Sobol sequences via indexes for generating minimally correlated hypervectors, particularly in applications related to symbol-oriented architectures. The performance of the proposed technique is evaluated in comparison to two traditional approaches of generating hypervectors based on linear-feedback shift registers and MATLAB random functions. The evaluation is conducted for three applications: (i) language, (ii) headline, and (iii) medical image classification. Our experimental results demonstrate accuracy improvements of up to 10.79%, depending on the vector size. Additionally, the proposed encoding hardware exhibits reduced energy consumption and a superior area-delay product.**

***Index Terms*—hyperdimensional computing, language processing, optimization, Sobol sequences, stochastic computing.**

I. Introduction

HYPERDIMENSIONAL computing (HDC) [1]–[3] is a trending paradigm that mimics important brain functionalities toward high-efficiency and noise-tolerant computation. The paradigm has shown significant promise for efficient and robust learning [4]. HDC can transform data into knowledge at a very low cost and with better or comparable accuracy to state-of-the-art methods for diverse learning and cognitive applications [5], [6]. The fundamental units of computation in HDC are high-dimensional vectors or "hypervectors" (consisting of $+1$s and -1s, or logic-1s and logic-0s) constructed from raw signals using an encoding procedure (Fig. 1(a)). During training, HDC superimposes together the encodings of signal values to create a composite representation of a phenomenon of interest known as a "class hypervector" (Fig. 1(b)). In inference, the nearest neighbor search identifies an appropriate class for the encoded query hypervector (Fig. 1(c)). Hypervectors have dimensionality, D, often in the orders of thousands of dimensions. A hypervector has a distributed, holographic representation in which no dimension is more important than others. The hypervectors in an HDC system are generated to have almost *zero* similarity. Previous works targeted *near-orthogonal* hypervectors by generating random hypervectors with approximately the same number of $+1$s and -1s [7]–[10]. But the inherent randomness in these conventionally generated hypervectors can lead to poor performance, particularly with smaller Ds. Low classification accuracy is likely in cases with poor distribution and undesired similarity between hypervectors.

Sercan Aygun is with the School of Computing and Informatics, University of Louisiana at Lafayette, Lafayette, LA, 70503, USA. E-mail: sercan.aygun@louisiana.edu. M. Hassan Najafi is with the Electrical, Computer, and Systems Engineering Department at Case Western Reserve University, Cleveland, OH, 44106, USA. E-mail: mhassan.najafi@case.edu.

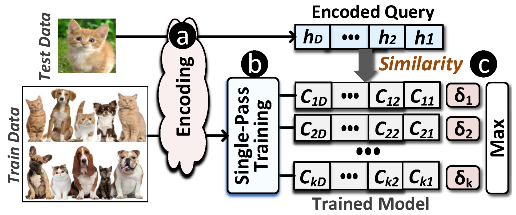

Test Data
Train Data
a
Encoding
b
Single-Pass Training
Encoded Query
h_D ••• h_2 h_1
Similarity
c
C_{1D} ••• C_{12} C_{11} δ_1
C_{2D} ••• C_{22} C_{21} δ_2
C_{kD} ••• C_{k2} C_{k1} δ_k
Max
Trained Model

Fig. 1. Classification with hyperdimensional computing.

Bit-stream computing, also known as stochastic computing (SC), has been the subject of a large body of recent research efforts due to attractive advantages such as very-low implementation cost and high tolerance to noise [11]. SC operates on random sequences of binary bits, called *bit-stream*. Similar to hypervectors in HDC, stochastic bit-streams are holographic with no bit significance. Complex arithmetic operations are simplified to basic logic operations in SC. For instance, multiplication can be performed using a single AND gate [12]. However, the accuracy of SC operations is severely affected by the random fluctuations in the bit-streams. Some operations, such as multiplication, similarity or correlation between bit-streams further degrades the quality of results. Often very long bit-streams need to be processed for accurate results. Recently, *low-discrepancy (LD)* bit-streams were suggested to improve the quality of SC operations while reducing the length of bit-streams [13]–[15]. Logic-1s and logic-0s are uniformly spaced in these bit-streams. Hence, the streams do not suffer from random fluctuations. The correlation issue is further addressed with these bit-streams by using different LD distributions. LD bit-streams have been recently used to perform completely accurate computations with SC logic [16]. The LD bit-streams are generated by using *quasi-random numbers* such as *Sobol* sequences.

This work takes advantage of LD sequences to improve the data encoding of HDC systems. To the best of our knowledge, this study is the first work to optimize Sobol sequences to enhance the orthogonality of hypervectors in HDC systems. Unlike the previous studies that use pseudo-random sequences for random vector generation, this work provides quasi-randomness and guarantees independence between hypervectors by using *optimized* LD Sobol sequences [17]. The primary distinction between *pseudo-randomness* and *quasi-randomness* lies in the deterministic, uniform, and fast converging distribution of numbers in LD sequences. Quasi-random sequences are designed to fill the space more uniformly without clustering, a common issue in pseudo-random sequences. This uniform distribution is particularly beneficial in high-dimensional spaces where pseudo-random sequences tend to leave large gaps. Moreover, quasi-random sequences can achieve faster convergence rates compared to pseudo-random sequences. Pseudo-randomness yields nearly random results and may produce different outcomes with each iteration, whereas recurring LD sequences exhibit deterministic behavior. In contrast to pseudo-random sequences that can only provide near-orthogonal hypervectors, quasi-random sequences can achieve ideal orthogonality and, thus, high accuracy in HDC systems. We first inspect the Sobol sequences obtained from the MATLAB tool. The hypervector representation in HDC is similar to the bit-stream representation in SC; hence, we utilize stochastic cross-correlation (SCC) [18], a metric used for determining the similarity of stochastic bit-streams, in evaluating hypervectors. We produce a matrix of hypervectors from Sobol sequences. The SCC metric is then used to measure the correlation of each hypervector pair, yielding a *distance matrix*. Any cell in the distance matrix shows the absolute deviation from the SCC=0, indicating independent hypervectors. We propose an algorithm to select the best independent Sobol-based hypervectors. We utilize the selected hypervectors for vector encoding in an HDC system case study. We apply our proposed scheme to language, headline, and medical image classification problems [6], [19], [20]. We further evaluate the hardware efficiency of the new encoding module for HDC systems. In summary, the main contributions are as follows:

- For the first time, we utilize optimized LD Sobol sequences in data encoding of HDC and unveil their potential performance.
- We propose an algorithm for selecting independent hypervectors by utilizing SCC metric.
- We find the top-performing Sobol sequences for generating independent hypervectors.
- We compare the performance of Sobol-based hypervectors with two traditional approaches of encoding hypervectors using 1) linear-feedback shift registers (LFSRs) and 2) the MATLAB random functions.
- Our experimental results show an accuracy improvement of up to 10.79% for text classification.
- Our new encoder module exhibits significant savings in energy consumption and area-delay product.

The rest of the paper is organized as follows: Section II presents some basic concepts of HDC and SC. Section III describes the proposed methodology and presents the optimization of the best independent sequence selection. Experimental results are presented in Section IV for the language, newspaper headline, and medical image classification problems. Potential impact and the future work of this study are discussed in Section V. Finally, Section VI concludes the paper.

II. Background

A. Hyperdimensional Computing (HDC)

HDC is a brain-inspired computational model based on the observation that the human brain operates on high-dimensional representations of data. Reasoning in this robust model of computation is done by measuring the similarity of hypervectors [21]. Hypervectors are D-dimensional sequences with $+1$ and -1 values (corresponding to *logic-1* and *logic-0* in hardware, respectively). Prior works on HDC target near-orthogonal hypervectors with random distribution and approximately the same number of $+1$ and -1. So, the threshold value (T) is set to 0.5 [6]. In HDC applications, using hypervectors with long lengths (in the range of 10,000 lengths or more) is common to reduce the similarity between the encoded vectors and improve the quality of results.

The basic operations in HDC are multiplication ($\oplus$: logical XOR), addition (Σ: bitwise population count), and permutation (Π: shifting). These operations are invertible and have linear time complexity. HDC systems first encode data with a proper technique according to the classification or cognitive tasks. Spatial, temporal, and histogram-based encoding techniques are used in the literature [22]. Encoders are divided into (i) record-based and (ii) n-gram-based approaches [23], [24]. The record-based approaches assign level hypervectors ($\boldsymbol{L}$, e.g., pixel intensity values in image processing application) and position hypervectors ($\boldsymbol{P}$, e.g., randomly generated vectors for pixel positions). Feature positions on data are encoded via $\boldsymbol{P}$s that are orthogonal to each other. On the contrary, level hypervectors are expected to have correlations between neighbors. The final hypervector is denoted as $\boldsymbol{\mathcal{H}} = \Sigma_{i=1}^{N}(\boldsymbol{L}_i \oplus \boldsymbol{P}_i)$, where N is the feature size. In image processing case, the pixels are considered as features, and the result of XORing each position and level hypervector ($\boldsymbol{P} \times \boldsymbol{L}$) is accumulated

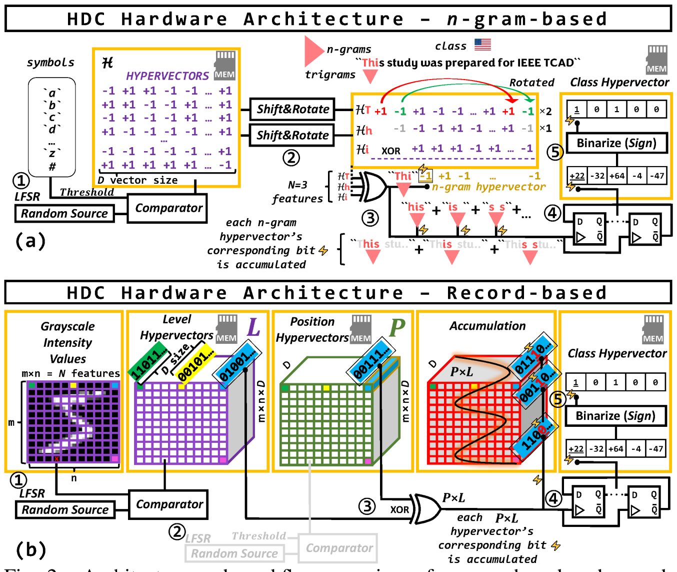


Fig. 2. Architecture and workflow overview of n-gram-based and record-based encoding for language processing and image classification. The steps involved in (a) n-gram-based and (b) record-based encoding methodologies.

to have a final D-sized vector that represents the overall image. The accumulation is performed considering the position-wise bits of different $\boldsymbol{P} \times \boldsymbol{L}$ vectors. The second category utilizes n-gram-based statistics like those in natural language processing systems. These encoders use rotationally permuted hypervectors, which are orthogonal to each other. The final hypervector is $\boldsymbol{\mathcal{H}} = \boldsymbol{L}_1 \oplus \pi \boldsymbol{L}_2 \oplus \pi^{N-1}\boldsymbol{L}_N$, where π^n denotes the n-times rotationally permuted $\boldsymbol{L}$. All samples in the training dataset are evaluated for $\boldsymbol{\mathcal{H}}$, and each contributes (via accumulation) to the corresponding class hypervector, which is the *trained model* of the overall system. For instance, in the text processing case, each incoming n sub-group of characters yields a single vector after shift-rotate-XOR, which is then accumulated for each one-character-shift n-gram vector. The accumulation is performed considering the position-wise bits of each n-gram vector. During the inference, the test data is encoded ($\boldsymbol{h}$), and the similarity check is performed between each test query and the class hypervector [25]. In our encoding scheme with Sobol sequences, we utilize both n-gram- and record-based approaches.

Fig. 2 provides an overview of the architecture and workflow involved in n-gram-based (Fig. 2(a)) and record-based (Fig. 2(b)) encoding for language processing and image classification problems. These models are integral to our study, as we consider both flows and hardware modules involved. Fig. 2(a) illustrates threshold-based hypervector generation and an example of n=3-gram-based encoding. We use +1 to denote logic-1 and -1 to denote logic-0. After generating random numbers in step ① and comparing them with the *threshold* (T) per symbol, the hardware block in step ② applies simple shifts and rotates to ensure orthogonality in the resulting vector obtained from the hardware XOR gate in step ③. Each n-gram hypervector from XOR blocks contributes to the corresponding class hypervector via the accumulator in step ④, depicted with a Johnson counter here. Finally, after all class members finish contributing to the signed accumulations, step ⑤ conducts simple binarization by a trivial subtraction (or comparison) from a threshold.

Fig. 2(b) depicts two generators for converting symbolic positions and numerical pixels into hypervectors. The remaining hardware blocks are similar to Fig. 2(a): XOR between $\boldsymbol{P}$ and $\boldsymbol{L}$ in step ③, accumulations in step ④, and binarization in step ⑤. The primary distinction between the two approaches lies in memory occupation; symbol-only-based representation occupies relatively less memory due to the absence of one group of hypervectors: numerical-related level hypervectors ($\boldsymbol{L}$). Both approaches need multiple random number genera-

TABLE I
COMPARISON OF SC AND HDC

	SC	HDC
Atomic Building Block	Bit-stream [26] (size of N)	Hypervector [27] (size of D)
Data Representation	Unipolar or Bipolar Bit-streams [26] Unary Bit-streams [28] Low-Discrepancy Bit-streams [16]	Random Hypervectors [6]
Metric	Stochastic Cross-Correlation [18]	Cosine Similarity [6], [29] Dot Product [30] Hamming Distance [23] Overlap Coefficient [23]
Target Representation	Uncorrelated Bit-streams [18] Correlated Bit-streams [28]	Orthogonal Hypervectors [20]

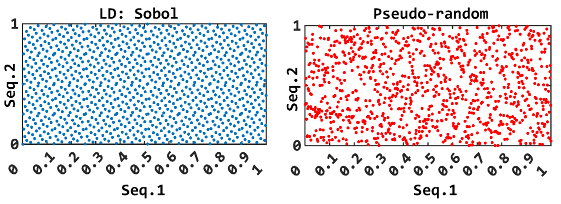


Fig. 3. Comparing the distribution of quasi-random Sobol and pseudo-random sequences.

tors. In this study, we utilize pre-ready sequences, eliminating the need for multiple random scalar trials, a departure from prior approaches.

B. Stochastic Computing (SC)

SC is a re-emerging paradigm that uses the power of processing random bit-streams to reduce the complexity of arithmetic operations to the level of individual logic gates [26], [31]. Let $X \in \mathbb{Z}_0^+$ be a scalar value to be represented with a stochastic bit-stream. A bit-stream $\boldsymbol{X}$ of size N has $P_X = \frac{X}{N}$ probability for the occurrence of 1s. Unlike conventional binary radix, stochastic bit-streams are free of bit significance. The ratio of the number of 1s to the length of bit-stream determines the bit-stream value. For instance, $\boldsymbol{X1} = 10101010$ represents $P_{X1} = \frac{4}{8}$ and $\boldsymbol{X2} = 10111101$ represents $P_{X2} = \frac{6}{8}$. Applying bit-wise AND operation to these bit-streams produces an output bit-stream $\boldsymbol{Y} = 10101000$ with probability $P_Y = \frac{3}{8}$ that is equal to $P_{X1} \times P_{X2}$. For correct functionality and accurate result, the two operand bit-streams need to be *independent* or *uncorrelated*. Accurate conversion of scalar values to stochastic bit-streams while guaranteeing independence between them has been a long-time challenge in SC [18]. The state-of-the-art work has addressed this challenge by encoding data to *LD bit-streams* [14], [31]. The input data is compared with quasi-random numbers such as Sobol numbers from a Sobol sequence (please see Section II-C and Appendix). The comparison output generates an LD bit-stream representing the input value. A 1 is generated if $scalar > random\ number$. A 0 is produced otherwise. For an N-bit long bit-stream, the input data is compared with N Sobol numbers. Table I provides a comparison between the SC and HDC computational models.

C. From Pseudo-randomness to LD: Sobol Sequences

Similar to SC, HDC systems require an accurate representation of uncorrelated vectors for high accuracy and random sources for hypervector generation. Randomness can be quantified by the occurrence of the numbers in the random sequence. Analyzing the distribution of the numbers, particularly among multiple random sources, gives interesting information regarding their orthogonality. The correlation between two random sources can be quantified by using metrics such as Hamming distance, dot product, or cosine similarity.

Pseudo-random sources rely on mathematical algorithms to produce sequences of numbers. Given the same initial condition (*seed*), the algorithm generates the same sequence of numbers. Well-known examples of pseudo-random algorithms

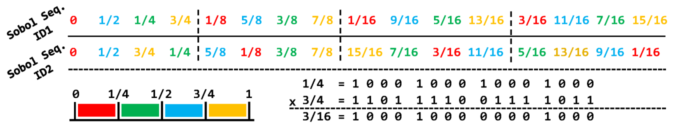


Fig. 4. Illustration of the first two Sobol sequences generated using MATLAB's `sobolset` function. The recurrence property and progressive precision are evident from the repeating and rotating patterns in binary vectors. Each vector contains subgroups with internal repetition or rotation.

are the Linear Congruential Generator, Mersenne Twister, and XOR-shift [32]. Quasi-randomness, on the other hand, holds a property of patterns that, while not genuinely random, possess certain characteristics of randomness. Quasi-random sequences distribute points more uniformly in space, exhibiting a more even distribution compared to pseudo-random sequences. Recurrence is another property of these sequences. A notable example of such sequences are Sobol or Halton sequences [33], [34]. Fig. 3 compares the distribution of quasi-random Sobol sequences and pseudo-random sequences. The x-y axes represent two different sequences.

This work employs Sobol sequences as the quasi-random source for HDC. Sobol sequences can be pre-generated using built-in algorithms in MATLAB tool [35] or Python packages [36]. These sequences exhibit LD properties, achieved through some basic computations on some "direction numbers" $(v1, v2, ...)$. Fig. 4 presents the first two Sobol sequences from the MATLAB tool. To show the recurrence property, we categorize consecutive groups of $\sqrt{D}$=2^2 numbers. When generating hypervectors, a threshold value (default 0.5) is compared with the sequence numbers. If the threshold value is greater or less than the sequence number, a 1 or 0 value is recorded, respectively. Fig. 4 shows examples of generating binary hypervectors for threshold values $1/4$ and $3/4$. With ideal orthogonality, a dot-product operation on the two hypervectors gives a result similar to the product of the threshold values. The example in Fig. 4 is an ideal case as the output vector represents $3/16$, which is equal to $1/4 \times 3/4$. We use a similar approach to show ideal orthogonality of quasi-random sequences over pseudo-random sequences. Table II reports the mean absolute error (MAE) results over 10,000 randomly sampled input pairs. As it can be seen, Sobol-based vector generation enables accurate multiplications with zero error. Conversely, when using LFSR-based pseudo-random sources, the MAE between the *obtained* and the *expected* product is non-zero, which exhibits inferior performance of pseudo-random sequences in generating orthogonal hypervectors.

The comparison between quasi- and pseudo-random sources can be extended to the distribution of Hamming distances between hypervectors. Fig. 5 illustrates the distribution of Hamming distances between pairs of hypervectors generated using Sobol sequences (Fig. 5(a)) and pseudo-random sources (binomial in Fig. 5(b) and LFSR in Fig. 5(c)). The distribution plots depict histograms of $10,000$ samples for generating hypervectors, with distance values represented on the x-axis and corresponding probabilities on the y-axis. Schmuck et al. [8] elucidate the use of Hamming distance as a metric for assessing orthogonality, where a score (d) ranging between 0 and 1 signifies the degree of orthogonality: d=0.5 indicates nearly orthogonal, d=0 correlated, and d=1 anti-correlated hypervectors.

In our experiment, we represent random scalar pairs as hypervectors. We generate $10,000$ pairs of random scalar values in the range of $[0, 1]$ (test samples) and convert them into hypervectors using both Sobol and pseudo-random sequences. We compare them based on the Hamming distance. For the Sobol case, we use the first two Sobol sequences from the MATLAB tool. For the second case, a binomial distribution, and for the third case, LFSRs serve as the random sources for generating D-dimension hypervectors. Upon comparing the three plots for D=128, we can see that values around 0.5 are more predominant in the Sobol-generated hypervectors

TABLE II
MAE OF SOBOL-BASED VS. LFSR-BASED PAIRS OF HYPERVECTORS FOR DOT-PRODUCT OPERATION

Pseudo-random (LFSR) MAE – D Vector Size											
(D=16)	(D=32)	(D=64)	(D=128)	(D=256)	(D=512)	(D=1,024)	(D=2,048)	(D=4,096)	(D=8,192)	(D=16,384)	(D=32,768)
0.0638	0.0308	0.0225	0.0142	0.0074	0.0035	0.0019	0.0012	5.95e-04	4.68e-04	2.11e-05	1.77e-05
Quasi-random (Sobol) MAE – D Vector Size											
(D=16)	(D=32)	(D=64)	(D=128)	(D=256)	(D=512)	(D=1,024)	(D=2,048)	(D=4,096)	(D=8,192)	(D=16,384)	(D=32,768)
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

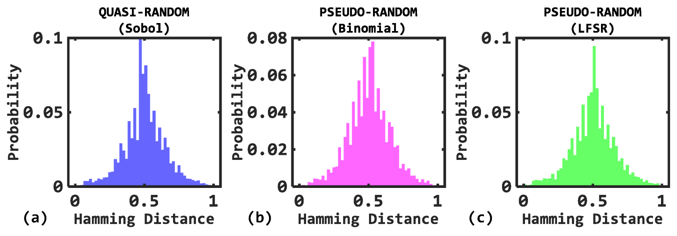


Fig. 5. Histogram of randomly sampled scalars' 1-to-all hypervector Hamming distances (d) with hypervector generation sources of (a) Sobol, (b) Binomial random MATLAB function, and (c) LFSR. D=128.

TABLE III
TOTAL POINTS (%) IN 1-VARIANCE DISTANCE FROM d=0.5 MEAN POINT

Hypervector (D)	Sobol LD	Pseudo-random (MATLAB)	Pseudo-random (LFSR)
D=128	**69.41%**	69.25%	69.17%
D=256	**69.70%**	69.67%	69.41%
D=512	**70.82%**	70.56%	70.60%
D=1,024	**70.13%**	70.11%	70.10%
D=2,048	**70.66%**	70.57%	70.60%
D=4,096	**71.55%**	71.25%	71.53%
D=8,192	**71.89%**	71.30%	71.65%
D=16,384	**72.32%**	72.16%	72.30%
D=32,768	**73.57%**	73.31%	73.50%

compared to those generated pseudo-randomly. We measured the percentage of values lying within one standard deviation of the mean (d=0.5), which serves as the orthogonality target. For the Sobol-based hypervectors, this percentage was 69.41%; for the binomial pseudo-random case, it was 69.25%, and for the LFSR, it was 69.17%. These results indicate a higher concentration of values around the target orthogonality point for the Sobol sequences. Table III provides the percentages of values lying within one standard deviation for varying D-sizes, further corroborating the effectiveness of Sobol sequences, especially for larger D values. The LD sequences consistently outperform pseudo-random ones in achieving orthogonality.

III. PROPOSED METHODOLOGY

A. From Bit-Streams to Hypervectors

SC and HDC both exploit a redundant *holographic* data representation. The *holographic* term refers to a mode of representation where information is distributed equally across all components. This ensures maximum robustness and efficient utilization of redundancy [2]. While conventional binary radix assigns weight to each bit depending on its significance, SC and HDC systems utilize unweighted sequences of binary bits [27], [37]. In both computational model, the encoding includes a comparison with a random value, R. A *scalar* value in SC and a *threshold* value in HDC are the actors of this comparison. Figs. 6(a) and (b) show examples of the traditional approaches for encoding data in SC and HDC. In Fig. 6(a), the scalar value X is encoded to a bit-stream of size $N = 8$ representing the probability P_X. The random source in Figs. 6(a) and (b) is a number generator that generates random numbers in the $[0, 1]$ interval. Random vector generation for HDC is shown in Fig. 6(b). Here, the threshold value (T) is 0.5. Fig. 6(c) shows how data is encoded to an LD bit-stream by exploiting a *Sobol* sequence.

The produced N-dimensional Sobol arrays with recurrence relation are used as an ideal random source to generate

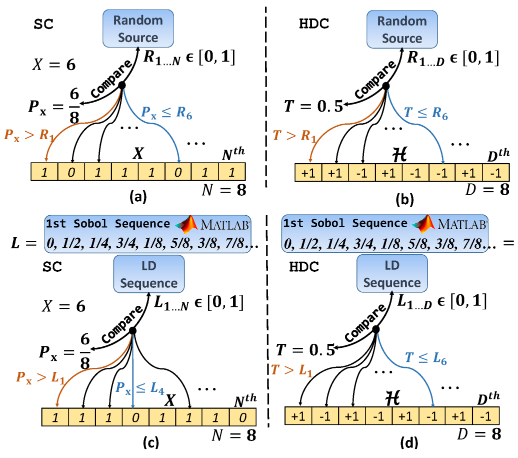


Fig. 6. (a) Conventional pseudo-random source-based stochastic bit-stream generation, (b) traditional hypervector generation using a pseudo-random source, (c) LD bit-stream generation using a Sobol sequence, and (d) utilizing Sobol sequences for generating hypervectors.

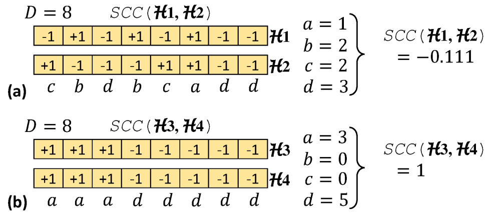

$D = 8$
$SCC(\mathcal{H}1, \mathcal{H}2)$
-1 +1 -1 +1 -1 +1 -1 -1 $\mathcal{H}1$
+1 -1 -1 -1 +1 +1 -1 -1 $\mathcal{H}2$
(a) c b d b c a d d
$a = 1$ $b = 2$ $c = 2$ $d = 3$
$SCC(\mathcal{H}1, \mathcal{H}2) = -0.111$
$D = 8$
$SCC(\mathcal{H}3, \mathcal{H}4)$
+1 +1 +1 -1 -1 -1 -1 -1 $\mathcal{H}3$
+1 +1 +1 -1 -1 -1 -1 -1 $\mathcal{H}4$
(b) a a a d d d d d
$a = 3$ $b = 0$ $c = 0$ $d = 5$
$SCC(\mathcal{H}3, \mathcal{H}4) = 1$

Fig. 7. Correlation measurement of two sample hypervectors: (a) near-zero correlation, (b) highly correlated.

accurate LD bit-streams. Sobol sequences have also been successful in providing the needed *independence* between stochastic bit-streams. Generating different LD bit-streams by using different Sobol sequences is sufficient to guarantee independence between bit-streams [16]. Motivated by the success of using quasi-random numbers in SC, this work employs Sobol sequences for generating hypervectors of HDC systems. Fig. 6(d) presents the idea. We use Sobol sequences as the random source to generate *uncorrelated* hypervectors with *desired ratio* of +1 and −1. Conventionally, HDC systems sets the *threshold* value (T) to 0.5 to generate hypervectors with 50% +1 and 50% −1. Unlike prior HDC systems, this work explores a range of values for T to achieve the best accuracy with Sobol-based hypervectors. The challenging optimization problem is to determine the best set of Sobol sequences for any T value. The merit metric for this optimization is SCC as given in equation (1):

$$SCC = \begin{cases} \frac{ad-bc}{D\times min(a+b,a+c)-(a+b)\times(a+c)} & , \; if \; ad > bc \\ \frac{ad-bc}{(a+b)\times(a+c)-D\times max(a-d,0)} & , \; else \end{cases} \quad (1)$$

The a, b, c, and d variables in the SCC equation [18] are the cumulative sum of overlaps between two hypervectors: $a = |\{\mathcal{H}x_i = \mathcal{H}y_i = +1\}|$, $b = |\{\mathcal{H}x_i = +1, \mathcal{H}y_i = -1\}|$, $c = |\{\mathcal{H}x_i = -1, \mathcal{H}y_i = +1\}|$, $d = |\{\mathcal{H}x_i = \mathcal{H}y_i = -1\}|$. SCC is a value in the $[-1, +1]$ interval. A zero or near-zero SCC means *uncorrelated* hypervectors. SCC=+1 indicates a positive correlation (totally similar), while SCC=−1 shows

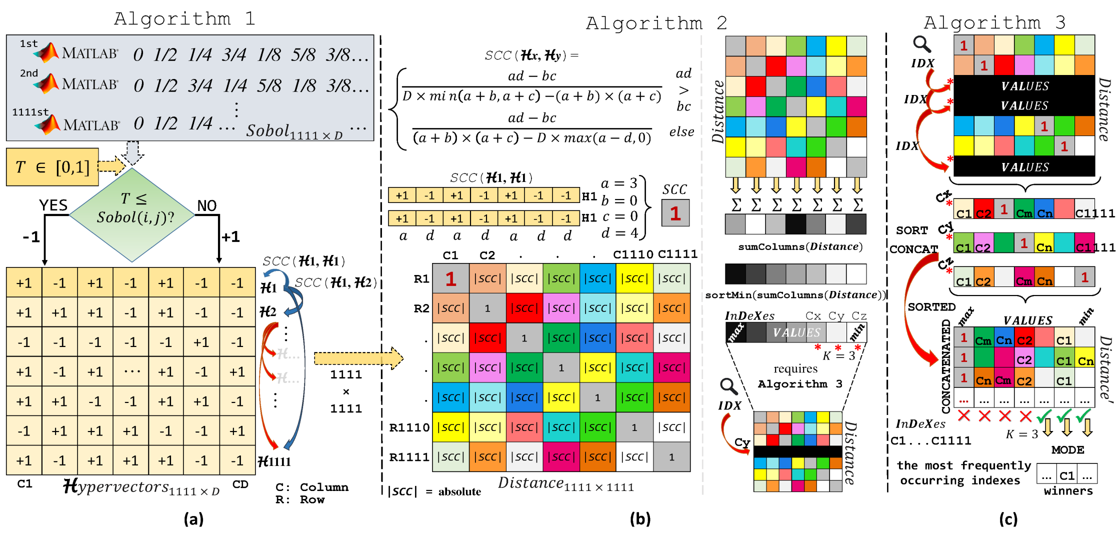


Fig. 8. The illustration of (a) Algorithm 1, (b) Algorithm 2, and (c) Algorithm 3.

Algorithm 1 Sobol-based $\mathcal{H}ypervectors$ Generation

```
Require: $Sobol_{1111\times D}$, $T$ : threshold, $D$ : $\mathcal{H}ypervectors$ size
Ensure: $\mathcal{H}ypervectors$
 1: for $i = 1 : 1 : 1111$ do
 2:     for $j = 1 : 1 : D$ do
 3:         if $T \le Sobol(i,j)$ then
 4:             $\mathcal{H}ypervectors(i,j) = -1$
 5:         else
 6:             $\mathcal{H}ypervectors(i,j) = +1$
 7:         end if
 8:     end for
 9: end for
10: return $\mathcal{H}ypervectors$
```

Algorithm 2 SCC of $\mathcal{H}ypervectors$ Cartesian Product

```
Require: $\mathcal{H}ypervectors$,
Ensure: $VAL$, $IDX$, $Distance$
 1: for $i = 1 : 1 : 1111$ do
 2:     for $j = 1 : 1 : 1111$ do
 3:         $Distance(i,j) \leftarrow$
 4:         $|\texttt{SCC}(\mathcal{H}ypervectors(i,:),\ \mathcal{H}ypervectors(j,:))|$
 5:     end for
 6: end for
 7: $[VAL,\ IDX] \leftarrow \texttt{sortMin}(\texttt{sumColumns}(Distance))$
 8: return $VAL, IDX, Distance$
```

a negative correlation (no overlap). Fig. 7 exemplifies two pairs of hypervectors and their corresponding SCC values. All hypervectors here have a probability $3/8$ of observing $+1$. The SCC value is calculated by finding the a, b, c, and d values. The example in Fig. 7(a) includes two hypervectors with near-zero SCC, while the hypervectors in Fig. 7(b) are identical and so have SCC=1.

B. On the Decision of the Best-Performing Sobol Sequences

Selection of the top-K best uncorrelated (orthogonal) sequences over m is a challenging problem and requires optimization. The complete heuristic solution space of this optimization has $P(m, K) = \frac{m!}{(m-K)!}$ different candidates (via permutation). In HDC, this problem turns out to be selecting the best orthogonal hypervectors in the pseudo- or quasi-random workspace. Algorithm 1 initiates the procedure for generating Sobol-based hypervectors considering the quasi-random space. We use the MATLAB tool and its built-in Sobol sequence generator [38], which implements Joe and Kuo's method [39] as discussed in the Appendix. The maximum number of Sobol sequences that MATLAB can produce is 1111. In Algorithm 1, D is the hypervector size. Thereby,

Algorithm 3 Minimum of Minima

```
Require: $IDX$, $Distance$, $K$
Ensure: $SobolUncorrelated$
 1: for $i = 1 : 1 : K$ do
 2:     $minOFmin \leftarrow \texttt{sortMin}(Distance(IDX(i),:))$
 3:     $Concatenated_{K\times D} \leftarrow \texttt{CONCAT}(minOFmin)$
 4: end for
 5: $SobolUncorrelated \leftarrow \texttt{MODE}(Concatenated_{K\times D}.IDX,\ K)$
 6: return $SobolUncorrelated$
```

the $Sobol$ matrix has a size of $1111 \times D$. All $Sobol$ numbers are compared with the T value, and the results (-1 or $+1$) are recorded in a $\mathcal{H}ypervectors$ matrix of $1111 \times D$ size.

Algorithm 1 returns the $\mathcal{H}ypervectors$ matrix. Algorithm 2 uses this matrix to generate an SCC-based $Distance$ matrix of Cartesian products. Each pair of $\mathcal{H}x$ and $\mathcal{H}y$ in $\mathcal{H}ypervectors$ is compared using the SCC metric, yielding the $Distance$ matrix size of 1111×1111 that holds the absolute values of SCC.

Lines 1 to 6 of Algorithm 2 build the $Distance$ matrix by calculating the SCC values. Any i^{th} row and j^{th} column intersection holds the correlation coefficient between the i^{th} and j^{th} hypervectors in the $\mathcal{H}ypervectors$ matrix. The $Distance$ matrix is a symmetric, square matrix holding $SCC = 1$s in the diagonal elements ($SCC(\mathcal{H}x, \mathcal{H}x) = 1$.) After producing the $Distance$ matrix, the algorithm performs column-wise operations to select the minimum distances at each column (similarly, row-wise operations are possible due to symmetry). Line 7 of Algorithm 2 first calculates the summation of each Distance matrix column, and then sorts the results of summations. The summation is a vector with 1×1111 elements. Minimum sorting is applied to find the values and indexes of Sobol elements with minimum SCC. Figs. 8(a) and (b) depict Algorithms 1 and 2, respectively. As it can be seen, after obtaining the $\mathcal{H}ypervectors$ matrix by comparing Sobol numbers with T, SCC measures the correlation of the Cartesian-based hypervector pairs. The $Distance$ matrix is symmetric, and each color represents the absolute value of an SCC. Fig. 8(b) shows how column-wise summation is performed in Algorithm 2, yielding symbolic gray-tone representation. After sorting, the *values* (VALs) and *indexes* (IDXs) are recorded. The top-K minimum values of these elements are called the minima of the $Distance$ matrix, and the corresponding IDXs are used for the *minimum of minima* in Algorithm 3, as depicted in Fig. 8(c).

Algorithm 3 returns a $SobolUncorrelated$ vector, which holds the K best uncorrelated Sobol indexes. The inputs of the algorithm are IDXs and the $Distance$ matrix from Algo-

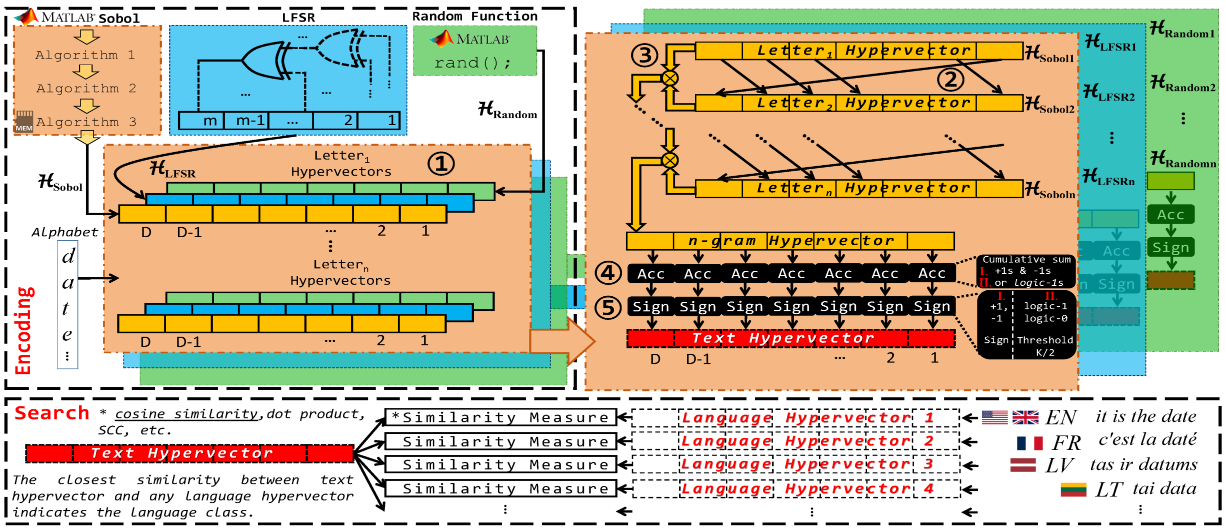


Fig. 9. The overall architecture of the HDC language classifier.

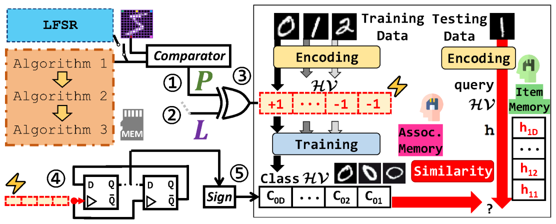


Fig. 10. The overall architecture of the HDC image classifier.

rithm 2, beside K, the number of to-be-selected Sobol indexes. The top-K minimum distances and their indexes (IDXs) from Algorithm 2 are used here. In Fig. 8(c), the selected IDXs of the $Distance$ matrix are pointed with black-colored rows. Each row is further processed for the minimum SCCs. The *for loop* in Algorithm 3 processes the K rows by sorting and CONCATaneting them, producing a new $Distance'$ matrix with a size of $K \times D$. Finally, the $Distance'$ matrix is checked with the MODE function to return the most repetitive IDXs of the top-K minimum-valued columns. Column-wise frequent index check allows us to see how repetitive Sobol elements are in the minima of SCC distances. Fig. 8(c) depicts Algorithm 3 for K=3; the top-K=3 minimum of minima are selected for the $SobolUncorrelated$ vector. If any repetitive IDX occurs, K is increased for the next available Sobol element to keep the column list unique in $SobolUncorrelated_{1\times K}$.

C. The Overall Architecture

In this section, we present the overall HDC architecture. Without loss of generality, we apply the proposed technique to a word-processing HDC system for language classification and an HDC system for image classification.

1) n-gram-based HDC Framework: In the first framework, a hypervector is needed for each alphabet letter. Our approach uses Sobol-based hypervectors. Nonetheless, we also implement the system with the MATLAB random function and an LFSR-based random number generator for comparison purposes. Sobol sequences are generated in a standard manner. Nevertheless, the selection of the best Sobol sequences is based on our optimization algorithms, which aim to achieve high orthogonality. The overall architecture is shown in Fig. 9. For each letter in the alphabet, hypervector generation is performed with our Sobol-based technique, LFSR, and the MATLAB built-in random function rand(), producing $\mathcal{H}_{Sobol}$, $\mathcal{H}_{LFSR}$, and $\mathcal{H}_{Random}$, respectively. The n-gram approach [23], is applied to the incoming data, i.e., n consecutive letters.

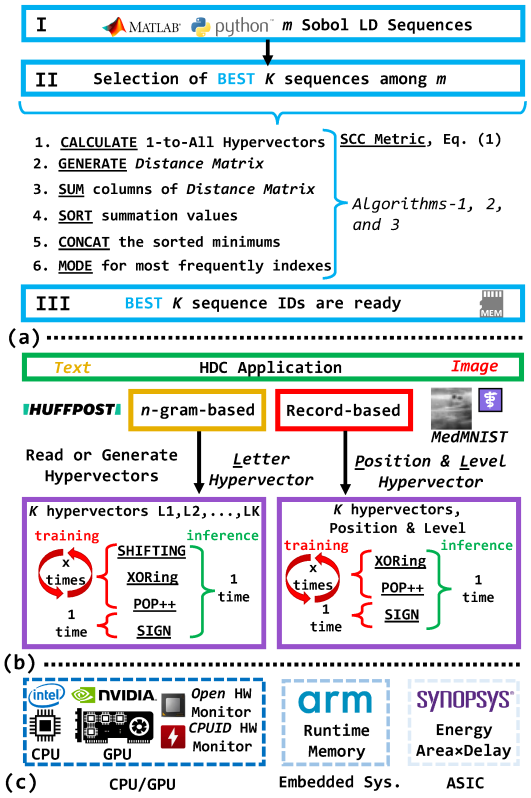

I MATLAB python m Sobol LD Sequences
II Selection of BEST K sequences among m
1. CALCULATE 1-to-All Hypervectors
2. GENERATE Distance Matrix
3. SUM columns of Distance Matrix
4. SORT summation values
5. CONCAT the sorted minimums
6. MODE for most frequently indexes
SCC Metric, Eq. (1)
Algorithms-1, 2, and 3
III BEST K sequence IDs are ready
(a)
Text HDC Application Image
HUFFPOST
n-gram-based
Record-based
MedMNIST
Read or Generate Hypervectors
Letter Hypervector
Position & Level Hypervector
training
inference
SHIFTING
XORing
POP++
SIGN
(b)
CPU
GPU
Open HW Monitor
CPUID HW Monitor
arm
Runtime Memory
SYNOPSYS
Energy Area×Delay
(c)
CPU/GPU
Embedded Sys.
ASIC

Fig. 11. (a) Overall flow of the proposed optimization steps for top-performing Sobol sequences. (b) Hardware details of the HDC architecture for the *training* and *inference* phases. (c) Tools utilized in the experiments.

For the LFSR-based approach, random numbers are generated using the maximal-period LFSRs described in [40] for each length D. The initial seed value of the LFSRs is randomly selected. Similar to Rahimi et al.'s HDC architecture in [6] and [41], the calculation of an n-gram hypervector is done by rotating the letter hypervectors, as shown in Fig. 9. We keep a copy of the hypervectors generated by the three approaches for this process. Then, the n-gram hypervector is accumulated.

During text hypervector generation, the *accumulation* operation (Acc) is an algebraic summation of +1s and −1s in hypervectors. The *thresholding* is applied via the sign function. In hardware, the population count of *logic-1*s is used

TABLE IV
HARDWARE EFFICIENCY CONSIDERING THE EMBEDDED PLATFORM, CPU, GPU, AND ASIC DESIGN

Performance		**(i) Performance in an Embedded Platform (ARM)**		**(ii) CPU Workload Performance (Intel i5)**		**(iii) GPU Power Load Performance (NVIDIA Quadro 6000)**		**(iv) Encoding Module ASIC Design (45 nm)**	
D	***HV Generation***	***Runtime***	***Memory***	***Average***	***Max.***	***Average***	***Max.***	***Energy***	***Area×Delay***
8192 n-gram	Pseudo-R.	1,068.3sec	18.3KB	10.6%(idle+9%)	15.2%(idle+13.2%)	0.354mW	0.402mW	16.88nJ	721.12$\times 10^{-9}$
	Sobol	687.4sec	17.8KB	9.4%(idle+7.4%)	12.1%(idle+10.9%)	0.036mW	0.256mW	2.69pJ	59.20$\times 10^{-12}$
8192 Rec.-bas.	Pseudo-R.	3,148.6sec	7.6KB	16.4%(idle+11%)	17.6%(idle+14%)	0.412mW	0.524mW	27.36nJ	1256.17$\times 10^{-9}$
	Sobol	1,948.5sec	5.2KB	14.4%(idle+10%)	15.7%(idle+12.3%)	0.051mW	0.334mW	4.04pJ	102.56$\times 10^{-12}$

for *accumulation*, followed by a *thresholding* operation by comparison with K/2 (K is the total number of the contributing n-gram hypervectors to the accumulator.) We proceed with the former approach in our simulations by using algebraic *accumulation* and sign-based *thresholding*. After iterating over the incoming data to generate the text hypervectors, the classification is performed in the search module by performing a similarity check between the text hypervector and the language hypervector.

2) Record-based HDC Framework: In the second framework, both *level* and *position* hypervectors are needed for pixel values and pixel positions, respectively. Fig. 10 illustrates the overall architecture of the HDC image classifier. We employ two random sources for comparison purposes: the LFSR method (*pseudo-random*), and the optimized Sobol sequences (*quasi-random*) using our proposed approach. Different from the prior art, better orthogonality is targeted during the intricate design of symbolic position hypervectors. Any random sequence, either pseudo- or quasi-random, is tested independently. Fig. 10 shows the overall hardware architecture, generating $\boldsymbol{P}$ and $\boldsymbol{L}$, XORing, accumulating result by D-type flip-flops, and binarization by *sign*. The hardware flow for each step aligns with Fig. 2. The training and testing methodologies are also illustrated in Fig. 10.

IV. TESTS AND RESULTS

This section details the performance evaluations of the proposed methodology. Fig. 11 shows the overall flow of the optimization steps and the HDC training and inference processes. Fig. 11(a) summarizes the optimization steps for identifying the best-orthogonal supplier Sobol sequences. Additionally, Fig. 11(b) presents a summary of operations and hardware for both training and inference, for image (record-based) and language-based (n-gram) applications. Notably, the cost of inference matches that of a single training attempt. The distinction lies in how each sample contributes independently to the class hypervectors during training, with binarization occurring only once all class samples are processed. Fig. 11(c) showcases the tools used and performance metrics considered.

A. Hardware Performance

We first evaluate the hardware efficiency of the proposed encoder module (all designs are in D=8192). We use four hardware workspaces: (i) ARM-based embedded platform (a resource-limited device with 700 MHz, 32-bit, single-core), (ii) Central processing unit (CPU - Intel(R) Core(TM) i5-10600K @4.10GHz), (iii) Graphics processing unit (GPU - NVIDIA Quadro RTX 6000), and (iv) and an ASIC design with 45 nm technology.

The first workspace considers two complete HDC systems with a pseudo-random-based approach and a Sobol-based approach. The overall system was implemented in C language and deployed into the ARM processor. The random method dynamically creates data with a built-in C language-based rand function. On the other hand, the Sobol sequences are pre-generated, stored in, and read from memory. Table IV presents the performance (i.e., run-time and memory usage) results. For the n-gram approach, the presented results are based on the single-time hypervector assignments for each letter; however, the training phase requires iteration for the random method that severely worsens during training. Record-based encoding for both $\boldsymbol{P}$ and $\boldsymbol{L}$ generations follow similar procedures. As the total symbol (pixel) count for the medical MNIST (medMNIST) [42] case study (28×28) is higher than the language case, relatively more runtime load is expected. However, compared to a 21-class language processing application, 8-class medMNIST (BloodMNIST) image classification requires relatively less model memory, as seen from the ARM-based memory occupation.

Table IV also shows the CPU-based workload performances of the encoding part of an HDC system. We iteratively (10^7 times) create an alphabet with letter hypervectors for language processing and also create the position and level hypervectors for image classification. We compare the workload brought by the random-based and Sobol-based approaches in the CPU. The iterations guarantee fairness in terms of possible background tasks and processes. We also put an idle time by waiting for stabilization before initiating each run. Table IV shows the average and maximum of all iterations. CPU workload during idle time and total increments on the hypervector operations (*idle + workload by vector generation* %) are recorded. The Sobol-based approach exhibits less load compared to the pseudo-random case in both letter-processing and image-processing problems. As images operate, both average and maximum loads brought in record-based encoding are higher than the n-gram case. We also report the GPU power consumption for the hypervector operations. As it can be seen, the Sobol-based approach provides nearly 10 times lower power consumption in both case studies.

Last but not least, we implement an ASIC hypervector encoding module to evaluate the energy and area-delay product. This design targets training-on-edge applications for future HDC frameworks, necessitating a comprehensive encoding module. For both n-gram- and record-based encoding architectures, corresponding to Fig. 2(a) and (b) (hardware in ①, ②, ③, ④, ⑤), we devised a design for the pseudo-random case comprising LFSR and Sobol sequences with recurrence property. Table IV reports energy consumption and area×delay. The Sobol approach yields significant outcomes in terms of energy consumption and area-delay product, promising advancements for next-generation of training-on-edge computing systems like HDC. Our proposed framework, leveraging deterministic Sobol sequences, frees us from dependence on random source generators such as LFSRs in training. It also reduces the optimization into a single step due to high-quality deterministic behavior. This framework enables us to maintain identical hardware steps for both training and inference (① - with memory read-, ②, ③, ④, and ⑤). In the conventional approach for the pseudo-random case, hypervectors are generated during training, and based on orthogonality performance, the hypervectors are saved for inference. While this remains an option for Sobol-based designs, thanks to their deterministic nature, it does not necessarily have "class counts × D size-bit memory." Instead, we save optimal Sobol sequences with 16-bit precision and can retain comparator-based dynamic hypervector generation even in inference, albeit with a trade-

TABLE V
CLASSIFICATION RATES FOR DIFFERENT ENCODING METHODS

D	**Encoding Methods**	**Min. Acc.**	**Max. Acc.**	**Std. Dev.**	**Avg. Acc.**
16	Random Vector	13.19%	18.14%	0.0108	15.07%
	LFSR w/ Random Seed	12.45%	18.20%	0.0119	15.17%
	Sobol LD Sequence		-		**17.80%**
	Sobol IDXs (T=0.66): 2, 3, 8, 11, 13, 14, 16, 18, 27, 28, 29, 34, 35, 40, 42, 44, 46, 47, 50, 52, 53, 54, 55, 60, 61, 62, 63, 66				
32	Random Vector	20.47%	25.70%	0.0121	22.71%
	LFSR w/ Random Seed	18.69%	25.70%	0.0121	22.47%
	Sobol LD Sequence		-		**26.52%**
	Sobol IDXs (T=0.30): 2, 4, 7, 8, 12, 20, 25, 27, 30, 29, 36, 40, 42, 39, 43, 58, 51, 52, 64, 66, 74, 46, 79, 69, 72, 80, 65, 88				
64	Random Vector	31.07%	39.52%	0.0151	35.43%
	LFSR w/ Random Seed	31.78%	37.10%	0.0124	34.57%
	Sobol LD Sequence		-		**46.22%**
	Sobol IDXs (T=0.70): 3, 6, 14, 16, 18, 19, 25, 26, 28, 30, 31, 35, 36, 41, 54, 49, 73, 61, 67, 64, 72, 90, 91, 86, 88, 96, 100, 101				
128	Random Vector	48.63%	54.80%	0.0127	52.04%
	LFSR w/ Random Seed	48.02%	54.02%	0.0112	51.16%
	Sobol LD Sequence		-		**60.74%**
	Sobol IDXs (T=0.74): 2, 4, 5, 7, 14, 11, 13, 16, 23, 24, 26, 25, 32, 34, 36, 42, 39, 53, 50, 48, 61, 67, 70, 64, 66, 72, 78, 84				
256	Random Vector	67.36%	71.70%	0.0103	69.24%
	LFSR w/ Random Seed	66.80%	71.67%	0.0105	68.61%
	Sobol LD Sequence		-		**79.03%**
	Sobol IDXs (T=0.70): 2, 3, 16, 10, 25, 28, 19, 38, 32, 60, 45, 49, 39, 53, 54, 68, 80, 82, 64, 90, 78, 95, 99, 112, 120, 108, 92, 118				
512	Random Vector	82.32%	83.83%	0.0039	83.03%
	LFSR w/ Random Seed	81.31%	83.18%	0.0045	82.22%
	Sobol LD Sequence		-		**89.47%**
	Sobol IDXs (T=0.70): 2, 8, 9, 13, 16, 19, 21, 28, 29, 48, 32, 44, 38, 64, 50, 51, 54, 58, 61, 85, 73, 74, 97, 101, 112, 90, 117, 120				
1024	Random Vector	90.27%	91.51%	0.0025	91.15%
	LFSR w/ Random Seed	90.01%	91.22%	0.0028	90.56%
	Sobol LD Sequence		-		**93.78%**
	Sobol IDXs (T=0.70): 1, 4, 9, 14, 26, 22, 21, 31, 51, 30, 34, 36, 61, 71, 67, 73, 97, 99, 100, 96, 107, 125, 108, 110, 152, 120, 150, 140				
2048	Random Vector	94.73%	95.40%	0.0015	95.14%
	LFSR w/ Random Seed	94.21%	95.04%	0.0017	94.71%
	Sobol LD Sequence		-		**96.31%**
	Sobol IDXs (T=0.34): 6, 5, 9, 14, 22, 16, 47, 51, 55, 64, 58, 68, 78, 87, 105, 88, 97, 96, 100, 166, 129, 132, 145, 152, 153, 114, 179, 164				
4096	Random Vector	96.68%	97.09%	9.20e-04	96.88%
	LFSR w/ Random Seed	96.44%	96.88%	0.0012	96.68%
	Sobol LD Sequence		-		**97.05%**
	Sobol IDXs (T=0.34): 1, 4, 6, 16, 30, 21, 26, 48, 33, 55, 43, 78, 60, 62, 82, 96, 97, 87, 88, 91, 92, 126, 100, 105, 109, 115, 121, 127				
8192	Random Vector	97.47%	97.87%	8.78e-04	97.68%
	LFSR w/ Random Seed	97.36%	97.73%	9.19e-04	97.55%
	Multiple LFSRs		-		97.31%
	Sobol LD Sequence		-		**97.85%**
	Sobol IDXs (T=0.38): 2, 5, 12, 15, 23, 36, 48, 51, 53, 54, 63, 73, 66, 79, 97, 115, 88, 98, 104, 109, 159, 148, 147, 123, 126, 130, 188, 172				

off in generation and memory cost, like in training. Table IV reports the cost of training a single data sample in ASIC design for the Sobol case. The reported results can also pertain to dynamically generated vectors in inference. Alternatively, we can read pre-generated binary vectors for $\boldsymbol{P}$ and $\boldsymbol{L}$ from memory to reduce the generation costs in inference. In this scenario, the area-delay product is found $18.32{\times}10^{-12}s \times m^2$ and $44.16{\times}10^{-12}s \times m^2$ for the letter and image processing applications, respectively. This binary vector reading approach is the same for both pseudo-random and Sobol case for the inference phase, while the Sobol-based design outperforms the pseudo-random case in the training phase as reported in Table IV.

B. Accuracy

For accuracy evaluation, we used two different datasets for separate performance monitoring: the 21-class European languages dataset [43], the newspaper headlines [44] dataset, and the medMNIST dataset [42]. Following the testing approach of [6] and [41], first, the Europarl Parallel Corpus dataset [45] was used for the inference step. The selected n-gram was four for better accuracy, as reported in [6]. First, the training dataset was pre-analyzed over a 1000-element validation set with the MATLAB tool's first 28 Sobol sequences. Hence, K=28 (26 letters, one space, and one extra character). Fig. 12 presents the pre-analysis for T. The T values on the x-axis range from 0 to 1 with 0.02 steps. The hypervector size D varies from 16 to 8192. The preliminary analysis for each D size helps us determine the approximate T range that gives the maximum accuracy with the Sobol sequences. We observed that, whether the first K Sobol sequences are used, or the K Sobol sequences are randomly selected, the relative analysis of the T values shows similar distributions, and the peak accuracy is obtained around T=0.34 ($\pm$0.04) or T=0.7 ($\pm$0.04). The selected Ts are used in Algorithm 1 with the best uncorrelated Sobol sequences obtained by Algorithms 2 and 3.

After determining the best T for each D, tests are performed with different encoding methods (Sobol, LFSR, and random). Table V presents the results. For the *LFSR*-based encoding, we evaluated all maximal-period LFSRs corresponding to each D [40]. For the *random* encoding, we run 1000 trials, each generating a different set of random numbers. We report the minimum, maximum, and average accuracy for the *LFSR* and *random* encoding. As it can be seen, the Sobol-based encoding achieves superior performance in all cases. The case with D=8192 delivers the best accuracy (97.85%), close to the baseline accuracy from a conventional machine learning approach [6]. The second-best outcomes are obtained with the hypervectors produced with the MATLAB *random* function, and then the hardware-friendly LFSR-based encoding provides the lowest accuracy. Table V reports the best Sobol IDXs selected with the proposed algorithms.

To show the superiority of the proposed encoding with another dataset, we also tested the classification problem of newspaper headlines on three topics (*entertainment*, *politics*, and *parenting*) obtained from the HuffPost newspaper headlines released in Kaggle [44]. This is a relatively more complex problem with shorter headlines compared to paragraphs. For training, 3400 headlines were utilized for each class, while 1000 different headlines were used for inference. Fig. 13 shows the pre-analysis of T for this dataset for D=4096 and 8192. As can be seen, a similar distribution is obtained by finding the best accuracy around T=0.34 ($\pm$0.04). For n-gram=5 and D=8192, the *random* function-based approach showed an accuracy of 69.81% (average of 1000 trials), while the proposed *Sobol*-based method achieved an accuracy of 70.97%.

A key observation considering the results presented in Table V and Figs. 12 and 13 is that utilizing Sobol sequences can achieve shorter hypervectors but with similar accuracy as their conventional pseudo-random counterparts. Consequently, iso-accuracy results can be produced with Sobol sequences with shorter delays, thereby achieving a more favorable area-delay product. These findings underscore the significance of the optimized Sobol-based architecture. The design is enhanced through optimization algorithms that leverage meticulously chosen, best orthogonal Sobol random sources. It should be emphasized that HDC systems generally adhere to a single-pass learning strategy, presenting results based on scanning the dataset only once without a backward pass. On the other hand, to establish edge-compatible training, the raw dataset has been presented without the use of any additional feature extraction or multiple iterations based on learning rate or error optimization. Conventional neural network systems with complex matrix multiplications pose bottlenecks for edge devices during error optimization with many partial derivative calculations, learning rate-based fine-tuning, and batch processing in multi-stochastic data processing. In contrast, HDC systems offer a lightweight solution for the same accuracy level. A neural network-based system established for similar accuracy is more costly and less efficient in hardware. The proposed vector generation technique opens a new perspective for hardware-friendly learning. It is lightweight, highly accurate, and requires only one round of hypervector generation. In prior HDC systems, selecting the best vectors involved multiple iterations with a random source to achieve good orthogonality [46]. In contrast, LD Sobol sequences inherently provide the needed orthogonality; with our optimization approach, higher quality is guaranteed for hypervector generation.

Following the n-gram-based HDC approach, we conducted experiments utilizing record-based encoding. In this set of

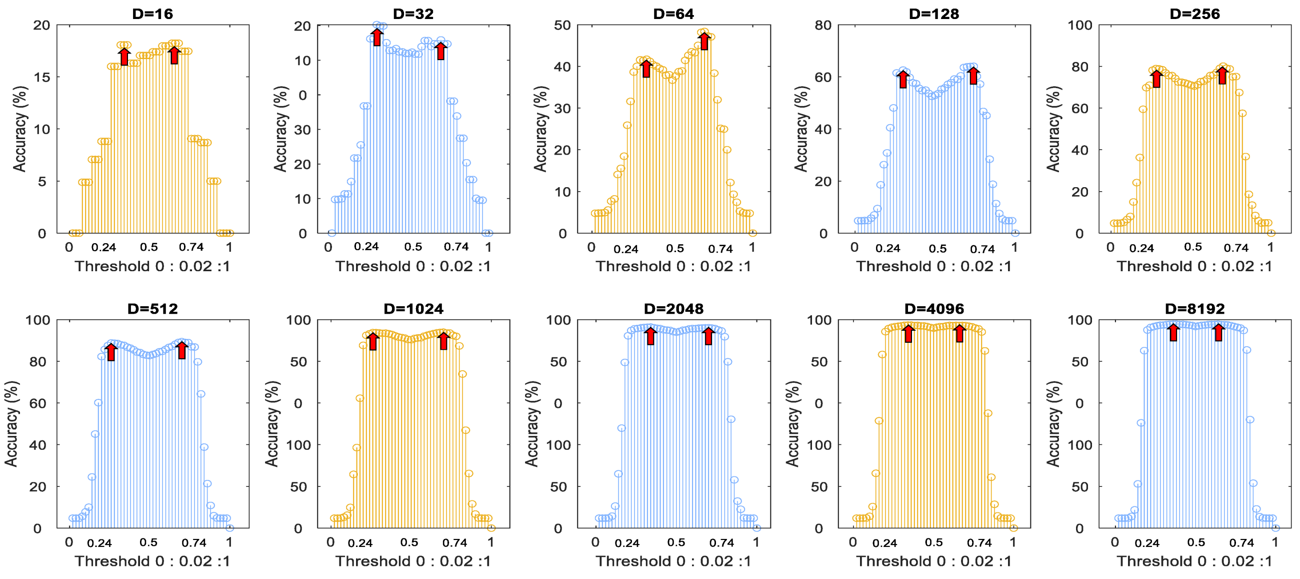
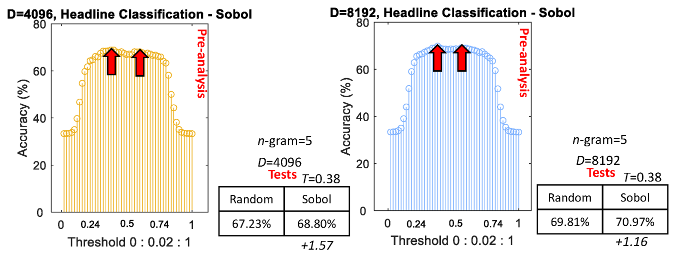


Fig. 12. Preliminary analysis of T for different hypervector sizes (D) with the first 28 Sobol sequences of MATLAB (🡅: maximum accuracy point).



Random	Sobol
67.23%	68.80%
	+1.57

Random	Sobol
69.81%	70.97%
	+1.16

Fig. 13. Preliminary analysis of T and tests for the headline dataset.

experiments, we also target measuring the learning rate-based training epochs for validations. We focus on two distinct sub-datasets for an image processing application using the medMNIST dataset: BloodMNIST and DermaMNIST. Our initial investigation involved validating our framework by employing optimized Sobol sequences for the record-based encoding of image pixel positions. Throughout this experiment, our primary objective was to monitor various learning metrics, particularly during training, given our proposal's alignment with a lightweight training system that involves less complexity and fewer iterations compared to the pseudo-random approach. We assessed the performance using a range of metrics based on confusion matrices, including Sensitivity ($\frac{TP}{(TP+FN)}$), Precision ($\frac{TP}{TP+FP}$), Specificity ($\frac{TN}{(FP+TN)}$), F1-measure ($\frac{2\times TP}{2\times TP+FP+FN}$), Balanced Accuracy ($\frac{Sensit.+Specif.}{2}$), and the Fowlkes–Mallows index (FMI: $\sqrt{(Precision \times Sensitivity)}$), in addition to standard accuracy ($\frac{TP+TN}{TP+TN+FP+FN}$, where TP: True Positives, TN: True Negatives, FP: False Positives, and FN: False Negatives). Both the pseudo-random (with LFSR) and Sobol-based approaches were evaluated in the experiments, and the validation accuracy, along with corresponding plots, was presented. The epoch-based learning is applied by adopting the learning rate (η)-based model update for incremental learning. The model undergoes validation accuracy evaluation during each training sample's process, considering the impact of the final contributing *training sample hypervector* (h). If validation improves based on the recent contribution to the class hypervector (C), then the training sample's effect on the learning model is incorporated, contributing to the class hypervector via accumulation (step ④ in Fig. 2). The formula for updating the class hypervector in the event of validation accuracy improvement is $C_{new} = C_{old} + (\eta \times h)$ (otherwise, $C_{new} = C_{old} - (\eta \times h)$ [47]). Our experiments achieved optimal results around η=0.1.

We separately tested two subsets of the medMNIST dataset: BloodMNIST, containing 17,092 images with 8 classes, and DermaMNIST, containing 10,015 images with 7 classes. Fig. 14 depicts the performance of the record-based HDC framework using pseudo-random and Sobol-based position encodings. We conducted 10 independent runs for the pseudo-random case (Fig. 14(a) for BloodMNIST and Fig. 14(c) for DermaMNIST) and reported the best possible learning plot and testing accuracy. The learning plots for epoch-based processing revealed the rapid convergence of validation accuracy in the Sobol case (Fig.s 14(b) and (d)). Moreover, the reported confusion matrices highlighted the superior performance of our proposed approach compared to conventional HDC learning methods. In the BloodMNIST dataset, while the accuracy for the pseudo-random case reached up to 83.96%, the Sobol-based approach exhibited improved performance, achieving an accuracy of 87.51%. In DermaMNIST, pseudo-random- and Sobol-based test accuracy is 67.66% and 70.34%, respectively; the DermaMNIST dataset is relatively challenging. These results highlight a notable trend: the Sobol-based approach exhibits higher validation accuracy in the initial epochs. For instance, by the tenth epoch, BloodMNIST achieves 90% validation accuracy with the Sobol approach, compared to around 80% with the pseudo-random approach. In essence, this observation may inspire future investigations into the training dynamics of HDC systems based on quasi-random sequences.

C. Iso-Accuracy (IA) Hardware Cost Analysis

Extending our examination of the hardware and accuracy results, we provide a deeper analysis of hardware performance for similar accuracy levels ("iso-accuracy") between the conventional random approach and the proposed optimization-based method. We find the accuracy of the random approach for the language classification task (recalling Table V) when setting D to 256, 512, 1024, 2048, and 4096. Subsequently, using optimized hypervectors based on the Sobol sequences in our method, we adjust the values of D to achieve a similar accuracy. For example, the random approach yields a classification accuracy of 68.6% with $D = 256$, while our method achieves a similar accuracy with $D = 185$. This provides a significant hardware cost advantage with our method. We assess various D sizes across different hardware

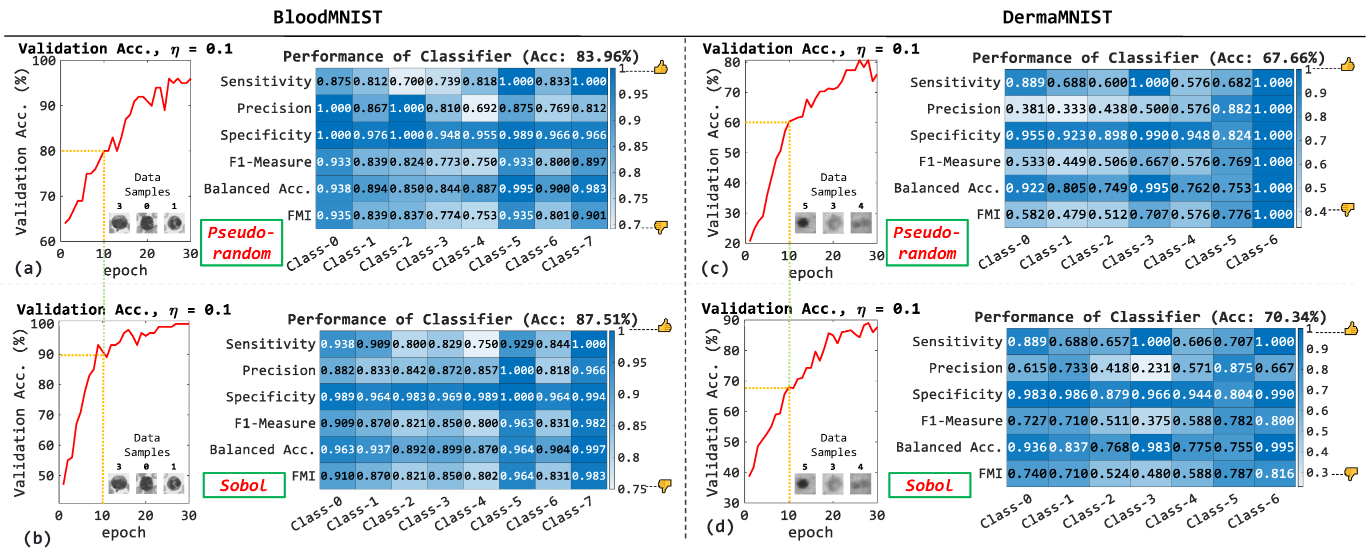


Fig. 14. Performance monitoring of medMNIST dataset with conventional pseudo-random LFSR and Sobol-based encoding approach. BloodMNIST dataset performance with (a) pseudo-random LFSR and (b) quasi-random Sobol sequences. DermaMNIST dataset performance with (c) pseudo-random LFSR and (d) quasi-random Sobol sequences.

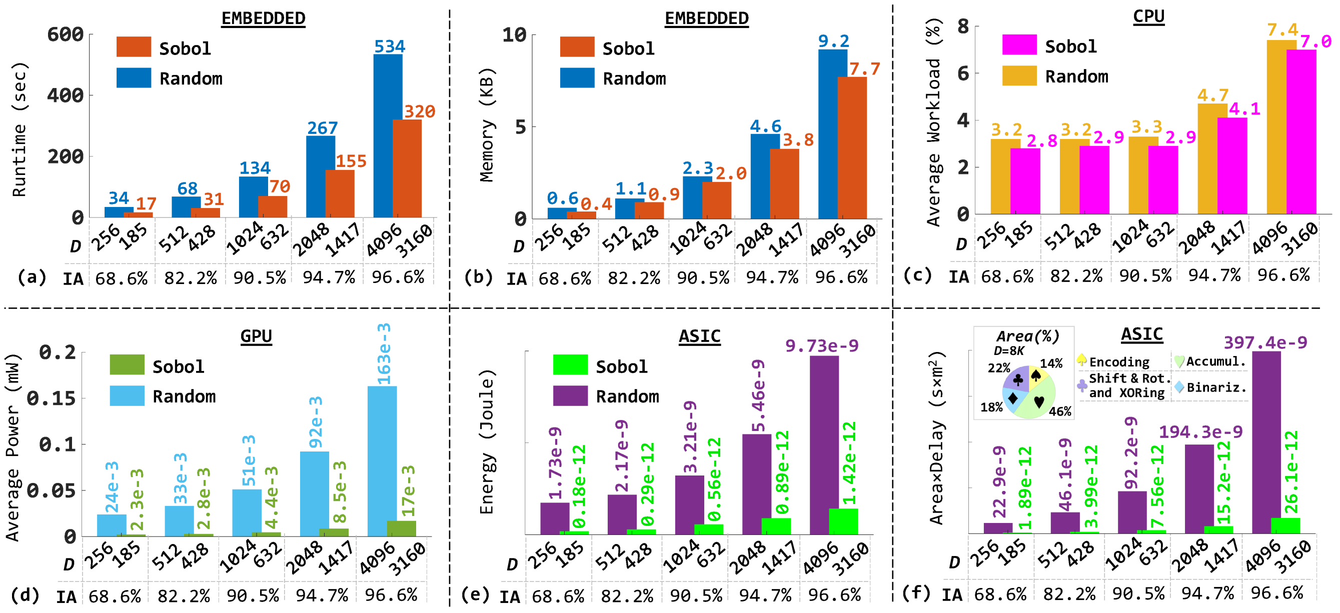


Fig. 15. Iso-Accuracy (IA) hardware cost analysis for parametric D values, (a) Embedded platform runtime, (b) Embedded platform memory, (c) CPU average workload, (d) GPU power, (e) ASIC energy, and (f) ASIC area×delay including the HDC sub-module area distribution given in a pie chart.

environments (i.e., an embedded system, a CPU, a GPU, and an ASIC design) to illustrate the trade-offs between accuracy and hardware resources. We also break down the hardware cost results into smaller components for the ASIC implementation. Fig. 15 depicts the results. As it can be seen, our proposed approach consistently exhibits better performance in terms of latency, power, energy, and memory consumption compared to the random approach. In particular, we report runtime and memory usage for an embedded platform (Fig. 15(a) and (b)), average CPU load (Fig. 15(c)), average GPU power consumption (Fig. 15(d)), and energy and area×delay for the ASIC design (Fig. 15(e) and (f)). We note that as the vector size or D increases, both the accuracy and the hardware cost increase. Finally, Fig. 15(f) illustrates a pie chart detailing the specific area consumption of each sub-module within our HDC system (for $D = 8192$ and n=3-gram), highlighting critical areas for potential optimization for further hardware cost reduction in future studies.

V. Potential Impact and Future Works

This study presents the first endeavor to optimize and select the best Low Discrepancy (LD) sequences for encoding hypervectors with maximum orthogonality for HDC systems. We expect this initiative forges new paths for future research on deterministic HDC systems, diverging from the reliance on stochastic pseudo-random encoding that necessitates continuous, time-consuming optimizations during the learning phase. Benchmarking HDC performance across different LD sources holds promise for stimulating new research avenues for improving the performance of HDC systems.

Recent literature has shown increasing interest in HDC system design for online training. Addressing the limitations associated with on-the-go selection of the best-performing random sequences, this study proposes a novel methodology

TABLE VI
METHODOLOGY CONTRAST: PROPOSED VS. RELATED WORKS

Features	Proposed Methodology	Literature
Sequence Generation	Quasi-random Sobol sequences with offline optimization for selecting the best sequence indexes for orthogonality	Pseudo-random sequences with relatively limited orthogonality properties
Training Algorithm	Sobol sequences with offline optimization for selecting the best sequence indexes for orthogonality	Pseudo-random sequences requiring multiple online runs of the learning algorithm for optimal performance
Adaptability	Pre-determined and valid for all future usage; no need for online training adjustments	Requirement for dynamic adaptation for optimal application performance
Distribution Patterns	Interesting, repeated pattern in the distribution of a sequence; easy inference of future distributions from early indexes	No estimable distribution, thus no assistance in inferring further sequence distributions from early indexes
Orthogonality for Training	Achieves satisfactory orthogonality for training learning systems through memory-saving	Requires additional circuitry for improved accuracy due to multiple iterations and dynamic nature
Accuracy and Learning Dynamics	Offers better accuracy and learning dynamics, including early epoch learning	Relatively lower accuracy and slower learning dynamics
Random Source for LD Sequence Index Selection	Utilizes stochastic computing-based random source for LD sequence index selection	Sequential usage of LD sequences (like in SC science)

for improved HDC system design by utilizing predefined deterministic sequences. Minimizing reruns during training can significantly alleviate the overall system load.

This study further creates new research avenues for applying HDC to larger learning tasks. One case is semantic analysis of linguistic data, like the headline dataset in this work. Addressing the challenges in raw symbol processing for HDC-based semantic classification, our proposed improvements hold promise for achieving high accuracy for tasks like large-scale language processing for semantics purposes. Our approach eliminates the need for application-specific online training optimizations, preserving on-the-go processing efficiency with a simple reading of pre-determined values from memory. While this study recommends quasi-randomness for higher accuracy, pseudo-random sequences (such as those generated using LFSRs) can also be explored to enhance the performance of prior work. Previous comparisons between Sobol- and LFSR-based encoding methods in the SC literature proved Sobol's superiority [16]. However, optimizing LFSRs for on-the-go hypervector generation may offer an intriguing alternative, albeit with potential accuracy trade-offs. Another future research direction of this work is to study scalability. The increase in the number of symbols to encode (e.g., in larger-scale image processing using more pixels) highlights the importance of vector orthogonality optimizations. Our proposed approach can find the best orthogonal symbol vectors for any space and K (symbol counts) size. In future work, we look into other types of quasi-random sequences, such as Van der Corput, Niederreiter, Halton, etc., the sequences already well acclaimed in SC [48], to increase the workspace for large-scale HDC applications with many symbols.

In addition to these impacts, this study bridges two emerging computing paradigms: HDC and Stochastic Computing (SC). Both HDC and SC enjoy holographic data representations. Improving the orthogonality for HDC and optimizing quasi-random sources can contribute to the accuracy of SC encoding. While SC traditionally employs two-input logic gates for arithmetic operations, exploring symbol-based representations can present novel opportunities for advancing this field.

Overall, this work provides a new path for HDC researchers to leverage quasi-randomness for ideal orthogonality in HDC systems. Exploring the encoding steps like binding and bundling for SC and hardware optimizations for better pseudo-random generators also present promising future directions. Large-scale HDC systems, especially for challenging language processing tasks, can significantly benefit from the proposed methodology by achieving higher accuracy and reducing vector size. In summary, Table VI outlines the key features of this work compared to the prevailing trends in the literature.

VI. CONCLUSION

In this work, we introduced a novel, lightweight approach featuring an innovative encoding technique for generating high-quality hypervectors for hyperdimensional computing (HDC) systems. Inspired by recent strides in low-discrepancy encoding methods proposed for stochastic computing (SC) systems, we employed quasi-random Sobol sequences, coupled with an optimization framework, to produce orthogonal hypervectors with varied distributions and ratios of +1s and -1s. Our methodology exploits an optimization algorithm to identify the optimal set of Sobol sequences, minimizing correlations crucial for vector symbolic data processing. To substantiate the effectiveness of the proposed technique, we conducted a comprehensive performance evaluation, comparing the method with two conventional approaches for hypervector generation based on LFSRs and algorithmic random functions. We evaluated the proposed approach for letter and image processing HDC systems, scrutinizing accuracy and integrating hardware designs across four distinct processing environments: ARM embedded device, CPU, GPU, and custom ASIC design. Our novel encoding technique demonstrated superior classification accuracy across varied datasets. It also showed higher hardware efficiency, considering factors such as energy efficiency and area-delay product.

ACKNOWLEDGMENTS

This work was supported in part by National Science Foundation (NSF) Grants 2339701 and 2019511, and generous gifts from Cisco and NVIDIA.

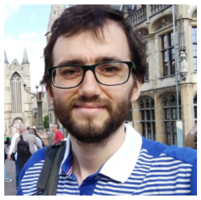
APPENDIX
FORMAL DEFINITION OF SOBOL SEQUENCES

The MATLAB built-in Sobol sequence generator [38] can be used to efficiently generate Sobol arrays. The procedure for generating Sobol numbers is as follows: *According to Joe and Kuo [39], any j^{th} component of the points in a Sobol sequence is generated by first defining a primitive polynomial $x^{s_j}+a_{1,j}x^{s_j-1}+...+a_{s_j-1,j}x+1$ of a degree of s_j in the field $\mathbb{Z}_2$. Any 'a' satisfies $a \in \{0,1\}$. By considering bit-by-bit XOR operator, $\oplus$, the 'a' coefficients are utilized for a sequence $\{m_{1,j}, m_{2,j}, ...\}$ by a relation given as $m_{k,j} = 2a_{1,j}m_{k-1,j} \oplus 2^2 a_{2,j}m_{k-2,j} \oplus ... \oplus 2^{s_j-1}a_{s_j-1,j}m_{k-s_j+1,j} \oplus 2^{s_j}m_{k-s_j,j} \oplus m_{k-s_j,j}$. The m values can be arbitrarily chosen provided that $1 \le k \le s_j$, and $m_{j,k} \in \{2n+1 : n \in \mathbb{Z}_0^+\}$ and $m_{j,k} < 2^k$. With a denote of direction numbers, $\{v_{1,j}, v_{2,j}, ...\}$, where any $v_{j,k} = \frac{m_{k,j}}{2^k}$, j^{th} component of the i^{th} point in a Sobol sequence is presented: $x_{i,j} = b_1 v_{1,j} \oplus b_2 v_{2,j} \oplus ...$, where any b is the right-most bits (i.e., least-significant ones) of the i sub-index in binary form.*

REFERENCES

[1] S. Aygun, M. S. Moghadam, M. H. Najafi, and M. Imani, "Learning from hypervectors: A survey on hypervector encoding," *arXiv preprint arXiv:2308.00685*, 2023.
[2] P. Kanerva, "Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors," *Cognitive Computation*, vol. 1, no. 2, pp. 139–159, 2009.
[3] M. Imani, Y. Kim, S. Riazi, J. Messerly, P. Liu, F. Koushanfar, and T. Rosing, "A framework for collaborative learning in secure high-dimensional space," in *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*, 2019, pp. 435–446.
[4] A. Hernández-Cano, C. Zhuo, X. Yin, and M. Imani, "Real-time and robust hyperdimensional classification," in *Proceedings of the 2021 on Great Lakes Symposium on VLSI*, ser. GLSVLSI '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 397–402. [Online]. Available: https://doi.org/10.1145/3453688.3461749
[5] A. Mitrokhin, P. Sutor, C. Fermüller, and Y. Aloimonos, "Learning sensorimotor control with neuromorphic sensors: Toward hyperdimensional active perception," *Science Robotics*, vol. 4, no. 30, p. eaaw6736, 2019.

[6] A. Rahimi, P. Kanerva, and J. M. Rabaey, "A robust and energy-efficient classifier using brain-inspired hyperdimensional computing," in *2016 International Symposium on Low Power Electronics and Design (ISLPED)*, 2016, p. 64–69.
[7] D. Ma, R. Thapa, and X. Jiao, "Molehd: Efficient drug discovery using brain inspired hyperdimensional computing," in *2022 IEEE International Conference on Bioinformatics and Biomedicine*, 2022, pp. 390–393.
[8] M. Schmuck, L. Benini, and A. Rahimi, "Hardware optimizations of dense binary hyperdimensional computing: Rematerialization of hypervectors, binarized bundling, and combinational associative memory," *J. Emerg. Technol. Comput. Syst.*, vol. 15, no. 4, oct 2019.
[9] F. R. Najafabadi, A. Rahimi, P. Kanerva, J. Han, and J. Rabaey, "Hyperdimensional computing for text classification," in *Design Automation Test in Europe*.
[10] M. Imani, X. Yin, J. Messerly, S. Gupta, M. Niemier, X. S. Hu, and T. Rosing, "Searchd: A memory-centric hyperdimensional computing with stochastic training," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 10, pp. 2422–2433, 2020.
[11] A. Alaghi and J. P. Hayes, "Survey of stochastic computing," *ACM Trans. Embed. Comput. Syst.*, vol. 12, no. 2s, may 2013. [Online]. Available: https://doi.org/10.1145/2465787.2465794
[12] S. Aygun, M. H. Najafi, M. Imani, and E. O. Gunes, "Agile simulation of stochastic computing image processing with contingency tables," *IEEE TCAD*, pp. 1–1, 2023.
[13] A. Alaghi and J. Hayes, "Fast and accurate computation using stochastic circuits," in *DATE'14*, March 2014, pp. 1–4.
[14] S. Liu and J. Han, "Toward energy-efficient stochastic circuits using parallel sobol sequences," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 7, 2018.
[15] S. Liu and J. Han, "Energy efficient stochastic computing with sobol sequences," in *2017 Design Automation and Test in Europe*, 2017, pp. 650–653.
[16] M. H. Najafi, D. J. Lilja, and M. Riedel, "Deterministic methods for stochastic computing using low-discrepancy sequences," in *2018 ICCAD*, 2018, p. 1–8.
[17] S. Aygun, M. H. Najafi, and M. Imani, "A linear-time, optimization-free, and edge device-compatible hypervector encoding," in *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2023, pp. 1–2.
[18] A. Alaghi and J. P. Hayes, "Exploiting correlation in stochastic circuit design," in *ICCD*, Asheville, NC, USA, 2013, pp. 39–46.
[19] G. Karunaratne, A. Rahimi, M. L. Gallo, G. Cherubini, and A. Sebastian, "Real-time language recognition using hyperdimensional computing on phase-change memory array," in *IEEE AICAS*, 2021.
[20] D. Kleyko, E. Osipov, and R. W. Gayler, "Recognizing permuted words with vector symbolic architectures: a cambridge test for machines," *Procedia Computer Science, 7th Annual International Conference on Biologically Inspired Cognitive Architectures*, vol. 88, 2016.
[21] P. Poduval, Z. Zou, H. Najafi, H. Homayoun, and M. Imani, "Stochd: Stochastic hyperdimensional system for efficient and robust learning from raw data," in *2021 Design Automation Conference*, 2021, pp. 1195–1200.
[22] A. Rahimi, P. Kanerva, L. Benini, and J. M. Rabaey, "Efficient biosignal processing using hyperdimensional computing: Network templates for combined learning and classification of exg signals," *Proceedings of the IEEE*, vol. 107, no. 1, pp. 123–143, 2019.
[23] L. Ge and K. K. Parhi, "Classification using hyperdimensional computing: A review," *IEEE Circ. and Syst. Mag.*, vol. 20, no. 2, pp. 30–47, 2020.
[24] Y. Yao, W. Liu, G. Zhang, W. Hu, and W. Xiong, "Fast sar image recognition via hyperdimensional computing using monogenic mapping," *IEEE Geo. and Rem. Sens. Let.*, vol. 19, pp. 1–5, 2022.
[25] H. Chen and M. Imani, "Density-aware parallel hyperdimensional genome sequence matching," in *2022 IEEE 30th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2022, pp. 1–4.
[26] A. Alaghi, W. Qian, and J. P. Hayes, "The promise and challenge of stochastic computing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 8, 2018.
[27] M. Imani, J. Messerly, F. Wu, W. Pi, and T. Rosing, "A binary learning framework for hyperdimensional computing," in *2019 Design Automation and Test in Europe*, 2019, pp. 126–131.
[28] P. Schober, M. H. Najafi, and N. TaheriNejad, "High-accuracy multiply-accumulate (mac) technique for unary stochastic computing," *IEEE Trans. on Comp.*, vol. 71, no. 6, 2022.
[29] P. Poduval, Z. Zou, X. Yin, E. Sadredini, and M. Imani, "Cognitive correlative encoding for genome sequence matching in hyperdimensional system," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*, 2021, pp. 781–786.
[30] S. Gupta, M. Imani, J. Sim, A. Huang, F. Wu, J. Kang, Y. Kim, and T. v. Rosing, "Cosmo: Computing with stochastic numbers in memory," *J. Emerg. Technol. Comput. Syst.*, vol. 18, no. 2, 2022.
[31] M. H. Najafi, D. Jenson, D. J. Lilja, and M. D. Riedel, "Performing stochastic computation deterministically," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 12, 2019.
[32] C. Dutang and D. Wuertz, "A note on random number generation," 06 2011.
[33] I. Sobol', "On the distribution of points in a cube and the approximate evaluation of integrals," *USSR Comp. Math. and Math. Phys.*, vol. 7, no. 4, 1967. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0041555367901449
[34] J. Halton, "On the efficiency of certain quasi-random sequences of points in evaluating multi-dimensional integrals." *Numerische Mathematik*, vol. 2, pp. 84–90, 1960. [Online]. Available: http://eudml.org/doc/131448
[35] sobolset, "Sobol quasirandom point set - MATLAB — mathworks.com," https://www.mathworks.com/help/stats/sobolset.html, 2024, [Accessed 27-March-2024].
[36] S. v1.13.0 Manual, 2024, [Accessed 27-March-2024]. [Online]. Available: https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.qmc.Sobol.html
[37] S. Aygun and E. O. Gunes, "Utilization of contingency tables in stochastic computing," *IEEE Trans. on Circ. and Syst. II: Expr. Br.*, vol. 69, no. 6, 2022.
[38] "MATLAB Sobolset," https://www.mathworks.com/help/stats/sobolset.html#brx24a7-6.
[39] S. Joe and F. Y. Kuo, "Remark on algorithm 659: Implementing sobol's quasirandom sequence generator," *ACM Trans. Math. Softw.*, vol. 29, no. 1, 2003.
[40] P. Koopman. Maximal length lfsr feedback terms. [Online]. Available: https://users.ece.cmu.edu/~koopman/lfsr/
[41] A. Rahimi, "Github," https://github.com/abbas-rahimi/HDC-Language-Recognition, 2016.
[42] J. Yang, R. Shi, D. Wei, Z. Liu, L. Zhao, B. Ke, H. Pfister, and B. Ni, "Medmnist v2 - a large-scale lightweight benchmark for 2d and 3d biomedical image classification," *Scientific Data*, vol. 10, no. 1, p. 41, Jan 2023.
[43] U. Quasthoff, M. Richter, and C. Biemann, "Corpus portal for search in monolingual corpora," in *LREC*, 2006.
[44] R. Misra, "News category dataset, kaggle," https://www.kaggle.com/datasets/rmisra/news-category-dataset, 2018.
[45] P. Koehn, "Europarl," http://www.statmt.org/europarl/, 2005.
[46] H. Lee, J. Kim, H. Chen, A. Zeira, N. Srinivasa, M. Imani, and Y. Kim, "Comprehensive integration of hyperdimensional computing with deep learning towards neuro-symbolic ai," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*, 2023, pp. 1–6.
[47] Y. Nam, M. Zhou, S. Gupta, G. De Micheli, R. Cammarota, C. Wilkerson, D. Micciancio, and T. Rosing, "Efficient machine learning on encrypted data using hyperdimensional computing," in *2023 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, 2023, pp. 1–6.
[48] M. Shoushtari Moghadam, S. Aygun, M. R. Alam, and M. H. Najafi, "P2LSG: Powers-of-2 low-discrepancy sequence generator for stochastic computing," in *Proceedings of the 29th Asia and South Pacific Design Automation Conference*, ser. ASPDAC '24. IEEE Press, 2024, p. 38–45. [Online]. Available: https://doi.org/10.1109/ASP-DAC58780.2024.10473928



Sercan Aygun (S'09-M'22) received a B.Sc. degree in Electrical & Electronics Engineering and a double major in Computer Engineering from Eskisehir Osmangazi University, Turkey, in 2013. He completed his M.Sc. degree in Electronics Engineering from Istanbul Technical University in 2015 and a second M.Sc. degree in Computer Engineering from Anadolu University in 2016. Dr. Aygun received his Ph.D. in Electronics Engineering from Istanbul Technical University in 2022. Dr. Aygun received the Best Scientific Research Award of the ACM SIGBED Student Research Competition (SRC) ESWEEK 2022, the Best Paper Award at GLSVLSI'23, and the Best Poster Award at GLSVLSI'24. Dr. Aygun's Ph.D. work was recognized with the Best Scientific Application Ph.D. Award by the Turkish Electronic Manufacturers Association. He is currently an Assistant Professor at the School of Computing and Informatics, University of Louisiana, LA, USA. He works on emerging computing technologies, including stochastic and hyperdimensional computing in computer vision and machine learning.

M. Hassan Najafi (S'15-M'18-SM'23) received the B.Sc. degree in Computer Engineering from the University of Isfahan, Iran, the M.Sc. degree in Computer Architecture from the University of Tehran, Iran, and the Ph.D. degree in Electrical Engineering from the University of Minnesota, Twin Cities, USA, in 2011, 2014, and 2018, respectively. He is currently an Assistant Professor with the ECSE Department of Case Western Reserve University (CWRU). Before that, he was an Assistant Professor at the School of Computing and Informatics at the University of Louisiana at Lafayette. His research interests include stochastic and approximate computing, unary processing, in-memory computing, and hyperdimensional computing. He has authored/co-authored more than 75 peer-reviewed papers and has been granted 5 U.S. patents with more pending. In recognition of his research, he received the NSF CAREER Award in 2024, the 2018 EDAA Outstanding Dissertation Award, the Doctoral Dissertation Fellowship from the University of Minnesota, the Best Paper Award at ICCD'17 and GLSVLSI'23, and the Best Poster Award at GLSVLSI'24. Dr. Najafi has been an editor for the IEEE Journal on Emerging and Selected Topics in Circuits and Systems, and a reviewer for many journals and conferences.