Independent and Dynamic Vector Symbolic Architecture for Hardware-Efficient Edge AI

Mehran Moghadam, *Graduate Student Member, IEEE*, Abu Masum, *Graduate Student Member, IEEE*, Sercan Aygun, *Senior Member, IEEE*, and M. Hassan Najafi, *Senior Member, IEEE*

***Abstract*—Hyperdimensional computing (HDC), also known as vector symbolic architecture (VSA), is a brain-inspired paradigm offering lightweight and hardware-efficient cognitive learning. By encoding data into high-dimensional hypervectors (HVs), HDC supports single-pass training and inherent robustness, making it highly attractive for edge AI. Yet, two challenges impede its deployment: efficient on-chip generation of orthogonal HVs and adaptation to dynamic data sizes without costly retraining. This work introduces the Independent and Dynamic VSA (`ID-VSA`), which advances HDC through five key innovations. First, we propose a compact single-source HV generator based on low-discrepancy (LD) sequences, enabling orthogonal symbol vectors with minimal hardware cost. Second, we present *Gaussian Polygon*, a multiscale learning mechanism that performs Gaussian-like interpolation directly in the HV domain. Third, we extend HV generation to quasi-normal distributions (QNDs), supporting both symbol and level vectors from the same randomness source. Fourth, we incorporate true-random number generation to exploit device-level noise for unbiased HV creation. Finally, we demonstrate flexible multi-assignment encoding for efficient *n*-gram processing. Evaluations demonstrate the proposed methods achieve accuracy improvements of up to 1.07% and 2.50% for image datasets MNIST and Pneumonia MNIST, and 18.36% on the language dataset over conventional HDC models. For larger-scale workloads, the proposed *Gaussian Polygon*-based designs achieve up to 4.33% improvement on the EuroSAT remote sensing dataset and up to 0.71% improvement on FractureMNIST3D medical dataset. Hardware synthesis in 45 nm technology confirms efficiency, achieving up to 370× lower power and 109× smaller area, establishing `ID-VSA` as a scalable solution for real-time, hardware-efficient edge AI.**

***Index Terms*—Edge AI, Gaussian pyramid, hyperdimensional computing (HDC), stochastic computing, vector symbolic architecture (VSA).**

I. Introduction

HYPERDIMENSIONAL computing (HDC) has attracted increasing attention in recent years due to its biologically inspired framework and reliance on associative and item memory models [2], [3], [4], [5], [6]. HDC enables the design of lightweight classification systems that map efficiently to digital hardware, making it particularly attractive for energy-constrained platforms. A key advantage of HDC is the ability to learn in a single pass or within just a few iterations, thereby avoiding the costly processes of backpropagation and gradient-based optimization. This makes HDC an excellent candidate for edge intelligence, where data are often incomplete, streaming, or noisy [7]. In HDC, information is encoded into high-dimensional random binary vectors in which all bits contribute equally. Such a representation ensures robustness against device-level faults, including stuck-at errors and soft bit flips, while still being capable of encoding both symbolic attributes (e.g., letters, positions, and timestamps) and numerical values [8].

A central challenge in realizing HDC at the edge is to develop autonomous, hardware-native mechanisms for hypervector (HV) generation that do not depend on pre-computed tables or software pre-processing. Traditional approaches typically rely on storing pregenerated HVs in memory [9], [10]. While this strategy may be acceptable for small-scale, fixed datasets, it becomes a bottleneck in real-world deployments where the set of symbols or features may expand or evolve over time. In such scenarios, scalability is fundamentally constrained by the available on-chip memory, and accommodating new symbols often requires over-provisioning memory resources upfront. In contrast, dynamic on-chip HV generation removes this dependency on static memory allocation and enables scalable, adaptive operation. By generating HVs on demand, the architecture can naturally support streaming and evolving data without incurring the substantial area and power

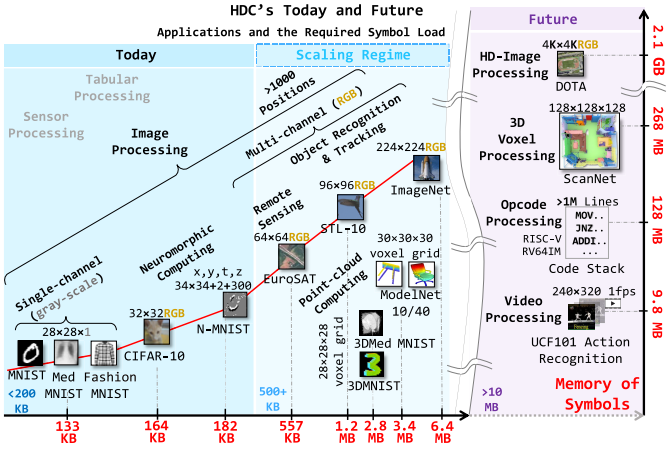

Fig. 1. Scaling of HDC symbol requirements and memory footprint: from simple image datasets to complex workloads.

Received 22 September 2025; revised 4 January 2026 and 23 February 2026; accepted 28 February 2026. Date of publication 16 April 2026; date of current version 28 April 2026. This work was supported in part by NSF under Grant 2019511, Grant 2339701, and Grant 2609436; in part by National Aeronautics and Space Administration (NASA) under Grant 80NSSC25C0335; in part by Vernon and Ruby Langlinais Non-Endowed Research Fund; in part by the Lockheed Martin Corporation Endowed Professorship Fund; in part by the Gifts from NVIDIA and Google; and in part by the NASA Award. An earlier version of this paper was presented at the IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED), 2025 [1]. *(Corresponding author: Sercan Aygun.)*

Mehran Moghadam and M. Hassan Najafi are with the ECSE Department, Case Western Reserve University, Cleveland, OH 44106 USA (e-mail: moghadam@case.edu; najafi@case.edu).

Abu Masum and Sercan Aygun are with the School of Computing and Informatics, University of Louisiana at Lafayette, Lafayette, LA 70503 USA (e-mail: c00591145@louisiana.edu; sercan.aygun@louisiana.edu).

Digital Object Identifier 10.1109/TVLSI.2026.3671174

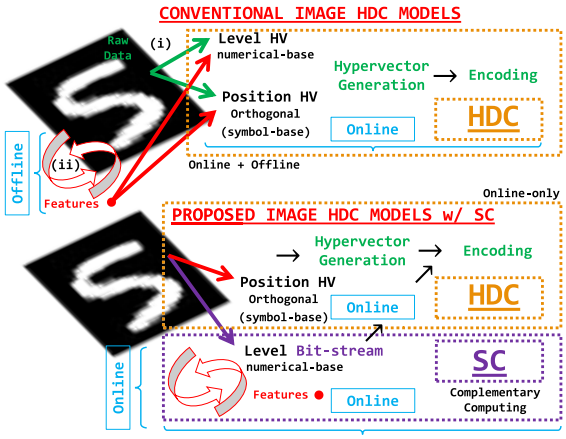

Fig. 2. Comparison of conventional and proposed **ID-VSA** models.

overheads associated with large memory structures, making it far better suited for resource-constrained edge platforms.

Fig. 1 motivates our adaptive on-chip HV generation by illustrating how the required number of symbols (e.g., positions and value/level elements) and the associated symbol-memory footprint rapidly grow from small grayscale images to RGB, video, and volumetric 3-D inputs. To quantify this trend, we use the symbol memory assuming a common HDC setting with a vector size of 1024 bits, and report the memory required for the corresponding symbol HVs under each workload per sample. Specifically, for each dataset application, we compute the number of distinct position symbols (e.g., *row* × *column* for images, *slice size* of 3-D data, or the *token* count for microprocessor opcode streams) and add the number of value (or level) symbols (e.g., 256 levels for 8-bit intensities), while treating RGB channels as feature channels rather than expanding them into separate position symbols. The resulting footprint remains below 200 KB for today's small-scale datasets (when no feature-based pre-processing is applied), but quickly increases to the MB range as resolution and modality scale up, and can reach tens of MB to a few GB for high-definition, data-centric workloads for futuristic HDC applications. This scaling makes dictionary-based storage and retrieval increasingly impractical for edge devices, and directly motivates dynamic and deterministic HV generation without large pre-stored tables.

To ensure reliable classification accuracy, each newly generated HV must remain orthogonal to all previously assigned symbols, thus preserving the symbolic separability that underpins HDC's performance. However, when vectors are generated randomly, this introduces an optimization challenge, as *near*-orthogonality alone does not guarantee maximal accuracy. To address this limitation, this work proposes a lightweight, fully on-chip vector generator capable of producing high-quality orthogonal binary vectors without requiring any postprocessing or iterative optimization. This design enables scalable, online, and adaptive learning directly on edge devices with a data-size agnostic approach, while maintaining guaranteed orthogonality through tight integration with *stochastic computing* (SC) [11], an emerging and complementary computing paradigm.

SC provides complementary benefits that can further enrich vector-symbolic learning [8]. SC also uses streams of randomly generated binary values for computation and representing information. In this work, we exploit SC to enhance the adaptability and learning dynamics of HDC. Instead of conventional retraining, we introduce `Gaussian Polygon`, a new interpolation-inspired learning approach modeled after the `Gaussian Pyramid` concept [12]. This scheme achieves Gaussian-like upscaling directly in the high-dimensional binary space, and is implemented through a low-cost stochastic logic network composed primarily of multiplexers (`MUX`s), building upon prior SC interpolation designs [13]. Beyond its computational efficiency, this approach directly highlights one of the persistent challenges in HDC: adapting to input data of variable sizes while maintaining accuracy. Fig. 2 contrasts conventional image processing pipelines in HDC with our proposed independent and dynamic vector symbolic architecture (**ID-VSA**). While the standard flow depends on offline feature extraction followed by encoding into HVs, our design leverages SC to construct HVs dynamically and directly from raw data, requiring no offline pre-processing. This creates a synergistic integration between SC and HDC: SC provides efficient bit-stream feature generation, while HDC consumes these enriched representations for robust learning at the edge.

Experimental results demonstrate that the proposed `Gaussian Polygon` learning approach significantly improves the learning dynamics of HDC, offering performance on par with retraining methods but at far lower hardware overhead. Evaluations demonstrate the proposed methods achieve accuracy improvements of up to 1.07% and 2.50% for image datasets MNIST and PneumoniaMNIST, and 18.36% on the language dataset compared to conventional HDC models. On larger-scale workloads, the proposed `Gaussian Polygon` based designs improve accuracy performance by up to 4.33% on EuroSAT and up to 0.71% over retraining baselines on 3-D FractureMNIST3D. Furthermore, our ASIC implementation in 45 nm technology demonstrates the efficiency of this design, consuming up to 370× less power and 109× less silicon area than baseline implementations. The key contributions of this work are as follows.

1) Introducing new orthogonal HV generation methods that enable HDC classifiers to operate dynamically at the edge, fully independent of fixed dataset sizes or pre-stored vectors.
2) Leveraging SC as a lightweight mechanism for online feature engineering, creating enriched HV representations directly from streaming data.
3) Moving beyond conventional retraining strategies by proposing `Gaussian Polygon`, an upscale learning approach that significantly reduces hardware overhead while sustaining high classification accuracy.
4) Employing true random number generation (TRNG) sources for HV generation in HDC systems, taking advantage of the inherently unbiased nature of physical randomness.
5) Enhancing HDC performance by introducing quasi-normal distribution (QND)-based HV generation, derived from a single quasi-normal random source.
6) Demonstrating robustness on larger-scale workloads with more symbol HVs in remote sensing and 3-D medical datasets.

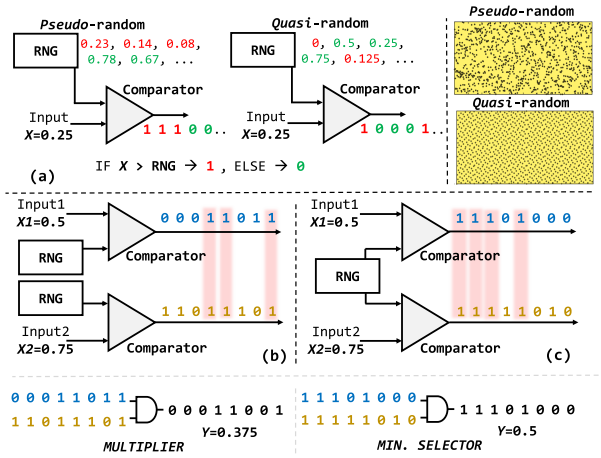

Fig. 3. (a) SC bit-stream generation, (b) *multiplication* by AND gate with uncorrelated inputs, and (c) *minimum* by AND gate with correlated inputs.

II. Background

A. Stochastic Computing

SC is a reemerging computing paradigm known for its low-cost and fault-tolerant hardware solutions [11]. Unlike conventional base-2 encoding, SC represents values using streams of randomly generated bits, where each bit has no positional significance. The value is determined by the probability of observing a logic-1 across the stream. For example, the value $X = 4/8$ can be represented by the bit-stream $\mathbf{X} = 10110010$ of length $N = 8$, or by any other bit-stream in which exactly half of the bits are "1," thereby preserving the same probability $4/8$.

Bit-stream generation in SC requires a random source and a comparator that compares the random numbers produced by the random source with the target value. A random number generator (RNG) produces numbers in a specified range, e.g., the unit interval $[0, 1]$. Each generated number is compared against the target value $0 \leq X \leq 1$ to form the bit-stream. The quality of randomness directly affects accuracy in SC arithmetic, making bit-stream generation a critical step. Fig. 3(a) illustrates the bit-stream generation process. Two main types of random sources are common: pseudo-random [14] and quasi-random, also known as low-discrepancy (LD) [15] sequences. Pseudo-random numbers scatter irregularly, which may introduce random fluctuations and undesired cross-correlations between generated bit-streams, degrading performance. In contrast, LD sequences fill the interval more uniformly and converge more quickly to the target probability.

Correlation control plays a crucial role in SC as it can alter circuit behavior. For example, an AND gate operates as a multiplier when its inputs are uncorrelated or statistically independent. The same gate functions as a minimum operator when driven by maximally correlated bit-streams. Correlation depends on how the bit-streams are generated: sharing the same random source maximizes correlation, while using different RNGs produces uncorrelated (nearly orthogonal) sequences. Fig. 3(b) shows an example with independent sources: $\mathbf{X1}$ = 00011011 (encoding $X1 = 4/8$) and $\mathbf{X2}$ = 11011101 (encoding $X2 = 6/8$). Their bit-wise AND gives

$$\mathbf{Y} = 00011001 \begin{pmatrix} X1 \rightarrow & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & 0 & 1 & \mathbf{1} \\ X2 \rightarrow & 1 & 1 & 0 & \mathbf{1} & \mathbf{1} & 1 & 0 & \mathbf{1} \end{pmatrix}$$

corresponding to $Y = 3/8$. This aligns exactly with the product of the encoded probabilities: $P_{X1} \times P_{X2} = 4/8 \times 6/8$. This result holds provided that input bit-streams are statistically *uncorrelated*, i.e., independent. Fig. 3(c) illustrates the opposite case with maximal correlation, where the same RNG is shared. Here, the bit-streams $\mathbf{X1}$ and $\mathbf{X2}$ overlap heavily, and the AND operation simply outputs the minimum of the two inputs (here, $\mathbf{X1}$).

A widely used operation in SC is scaled addition, which can be realized using a simple 2-to-1 MUX [16]. Two input bit-streams ($\mathbf{X1}$ and $\mathbf{X2}$) are connected to the MUX inputs, while the select line S is driven by a third bit-stream that encodes a weighting factor (e.g., 0.5). The MUX output randomly forwards bits from inputs according to the probability carried by S. When $S = 0.5$, half of the output bits are drawn from $\mathbf{X1}$ and half from $\mathbf{X2}$, yielding their average value. In general, the MUX behavior can be expressed as: $P_Y = P_{X1}(1 - P_S) + P_{X2}P_S$. When P_S is 0.5, this simplifies to: $P_Y = (P_{X1} + P_{X2})/2$.

B. Hyperdimensional Computing

HDC, also known as vector symbolic architecture (VSA), is an emerging paradigm inspired by how the human brain processes information using high-dimensional patterns [17]. In HDC, data are encoded using HVs containing hundreds to thousands of bits. HVs can be binary (1/0 in hardware) or bipolar (+1 and −1 in software). This unconventional representation is highly tolerant to noise and well-suited for capturing complex structures in data [7]. As in SC, randomness plays a central role in HV generation. The core principle of HDC is to leverage the properties of high-dimensional spaces to hold and manipulate information in a distributed manner.

HDC encoding is based on three core operations: *binding*, *bundling*, and *positional encoding* [18]. In software (bipolar HVs), these correspond to *multiplication*, *addition*, and *permutation*. In hardware, these are efficiently mapped to XOR (binding), population count (bundling), and shift-and-rotate operations (positional encoding). Together, these operations provide a way to represent and process symbolic information in a form that resembles cognitive processing.

Symbolic representation in HDC requires a high level of orthogonality (i.e., low correlation) between distinct symbols (e.g., letters, DNA nucleotides, image pixel positions, signal timestamps, etc.), where the order of data in a sequence is critical. Ideal orthogonality is achieved when each HV has a balanced distribution of +1s and −1s (or logic-1s and 0s), resulting in minimal correlation between distinct symbols. Random generation of HVs provides *near* orthogonal vectors, enabling the system to easily distinguish between unique symbols. This idea mirrors the role of uncorrelated bit-streams in SC, making the two paradigms naturally complementary. For example, LD sequences explored for SC [15] can improve orthogonality in HDC vector generation. Since information is distributed across all dimensions, HVs also show robustness to errors, a property that aligns closely with SC's reliance on collective rather than individual bit significance [19].

Fig. 4 illustrates two common HDC encoding strategies: "n-gram encoding" for text [Fig. 4(a)] and "record-based

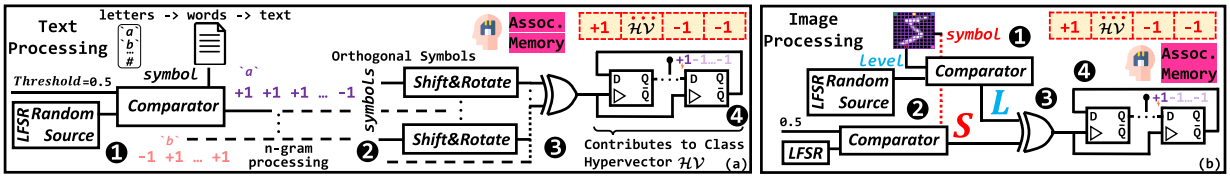

Fig. 4. The text and image processing pipelines in HDC. (a) Text processing: letters are converted into orthogonal symbol vectors via a pseudo-random source and a comparator, followed by n-gram encoding: *shifting-and-rotating*, XOR, and *accumulate* operations. (b) Image processing: pixel values are processed using intensity (L, level) and symbol-based position (S) vector encoding; Similar encoding steps are applied.

Algorithm 1 Query HV Encoding and Retraining in HDC

Input: Training set $\{(\vec{x}_i, y_i)\}_{i=1}^{N}$, class HVs $\{\vec{C}_k\}_{k=1}^{K}$, learning rate η, retraining epochs E
Output: Updated class HVs $\{\vec{C}_k\}$
for $e = 1$ to E **do**
  for $i = 1$ to N **do**
    $\vec{Q}_i \leftarrow \sum_j (\vec{S}_j \oplus \vec{L}_{ij})$ // *Query HV generation*
    $\hat{y}_i \leftarrow \arg\min_k$ Hamming$(\vec{Q}_i, \vec{C}_k)$ // *Hamming distance*
    if $\hat{y}_i \neq y_i$ **then** // *Retraining*
      $\vec{C}_{\hat{y}_i} \leftarrow \vec{C}_{\hat{y}_i} - \eta \vec{Q}_i$
      $\vec{C}_{y_i} \leftarrow \vec{C}_{y_i} + \eta \vec{Q}_i$
    end if
  end for
  for $k = 1$ to K **do**
    $\vec{C}_k \leftarrow \vec{C}_k / \|\vec{C}_k\|$
  end for
end for
return $\{\vec{C}_k\}$

encoding" for images [Fig. 4(b)]. The first step is vector generation. In text encoding [Fig. 4(a)], HVs are first assigned to letters and then combined using permutation, binding, and bundling. In image encoding [Fig. 4(b)], two types of HVs are produced: position vectors (S, symbolic) and intensity (or level) vectors (L, numerical). Symbolic HVs are generated by comparing random values against the corresponding positions (unbiased 0.5 probability value) across D cycles, while numerical HVs are produced proportionally to their values (recall bit-stream generation from Fig. 3). These HVs can be generated dynamically using linear-feedback shift registers (LFSRs) as a random source, or alternatively loaded from pre-stored HVs in memory [20]. After generation, encoding proceeds in three steps: *permutation* (Step ❷ in Fig. 4(a), used only in n-gram encoding), *binding* [Step ❸ in both Fig. 4(a) and (b)], and *bundling* (Step ❹ in both cases). *Permutation* improves vector orthogonality by shuffling vector elements (shifting-and-rotating) and preserving positional information for each symbol (e.g., no-shift, 1-shift, 2-shift, etc.) [2]. Although traditional HVs are generated using pseudo-random methods, these approaches can lead to low orthogonality and so poor performance [21]. *Binding* combines multiple HVs through multiplication or XOR. Finally, *bundling* merges multiple HVs into a single composite HV, maintaining semantic content while allowing holistic representation [2].

HDC performance can be improved through *retraining*, which performs multiple passes over the training data to refine the class HVs. During retraining, mispredicted queries are removed from the incorrectly predicted class HVs and added to the corresponding true class HV. Algorithm 1 summarizes the query HV encoding and the retraining procedure. Given an input sample $\vec{x}_i$, a query HV $\vec{Q}_i$ is constructed using record-based encoding. For each feature index j (e.g., pixel position), a symbol HV $\vec{S}_j$ encodes positional information, while a level HV $\vec{L}_{ij}$ represents the corresponding feature value. These two HVs are bound using the XOR operation and then bundled and accumulated across all features, $\vec{Q}_i = \sum_j (\vec{S}_j \oplus \vec{L}_{ij})$, resulting in a single high-dimensional representation of the input sample. During retraining, the query HV $\vec{Q}_i$ is compared against all class HVs $\{\vec{C}_k\}$ using a similarity metric. The predicted class label $\hat{y}_i$ is selected as the class with the highest similarity. If the predicted label differs from the true label y_i, retraining updates are applied by penalizing the predicted class HV and reinforcing the true class HV: $\vec{C}_{\hat{y}_i} \leftarrow \vec{C}_{\hat{y}_i} - \eta \vec{Q}_i$, $\vec{C}_{y_i} \leftarrow \vec{C}_{y_i} + \eta \vec{Q}_i$, where η denotes the optimal learning rate. This process is repeated for all training samples across multiple retraining epochs. At the end of each epoch, the class HVs are normalized to prevent uncontrolled growth and stabilize subsequent similarity computations. While retraining can improve classification accuracy, it introduces additional training-time overhead due to repeated query encoding, similarity evaluations across all classes, conditional HV updates, and normalization operations. The retraining process is applied only during the training phase, and inference remains unchanged, consisting of a single query encoding followed by similarity comparison using the final trained class HVs.

III. Proposed Methodology: ID-VSA

ID-VSA introduces five complementary design strategies to enhance the HDC system's functionality. Because HDC performance depends strongly on vector generation and encoding [21], we prioritize low-cost, high-quality vector generation to ensure the system remains independent and adaptable across tasks. We note that the "dynamic" capability of **ID-VSA** refers to supporting variable M (symbols/positions/tokens) without relying on a pre-stored dictionary, while maintaining a fixed HV dimensionality D for a given hardware instantiation. Our approach departs from the conventional *Store-and-Retrieve* model to a *Compute-on-Demand* pipeline. Rather than pre-generating and storing a full dictionary of symbol HVs, we generate the required HVs on the fly, symbol-by-symbol, using only a small temporary register. HV generation and encoding are overlapped in a hazard-free streaming manner; each generated HV is immediately consumed by the encoder, while the generator advances to the next symbol. We study five design variants that differ in their randomness sources, degree of determinism, and processing flow. Table I provides a general overview and comparison of the baseline HDC architecture and all proposed **ID-VSA** designs.

TABLE I
LANDSCAPE OF THE BASELINE HDC AND ID-VSA METHODS

HDC Design	Randomness Methodology	Purpose	Level HV source	Position/Symbol HV source	Orthogonality level	Key note
Baseline	Pseudo-randomness (conventional HDC)	Static fixed symbol set	LFSR/Counter + comparator (Uniform distribution)	Pre-stored in Look-Up Table (Uniform distribution)	Weak	Optional retraining (training-only), offline storage, *Non-deterministic*
ID-VSA ❶	Quasi-randomness (uniform) single-source VDC-2	Dynamic Symbol HV gen.	N/A (Symbols only)	VDC-2 + comparator Compute-on-demand	Strong	No-shift, Multi-assignment supported, *Deterministic*
ID-VSA ❷	Gaussian Polygon (utilizing SC)	Data pre-processing multi-scale training	Gaussian seq. + comparator 4-to-1 MUX for SC interpolation	Single-source (VDC-2)	Strong	Avoids retraining via multi-scale encoding, *Deterministic*
ID-VSA ❸	QND from bundled LD seq. (Sobol, VDC-2^n, hybrid)	Dynamic Symbol HV gen.	N/A (Symbols only)	Quasi-Normal HV ⊕ Uniform HV	Strong	Single-source QND can be prepared offline, *Deterministic*
ID-VSA ❹	QND from bundled LD seq. (symbol + level)	Dynamic Symbol HV gen. Data pre-processing	QND seq. + comparator	Quasi-Normal HV ⊕ Uniform HV	Strong	Unified source for symbols and levels, *Deterministic*
ID-VSA ❺	TRNG stream (Uniform distribution)	Dynamic Symbol HV gen.	N/A (pairs with **ID-VSA ❷** for images)	TRNG stream → HVs via block extraction or sliding window	Strong	Uniform-only, No QND support *Non-deterministic*

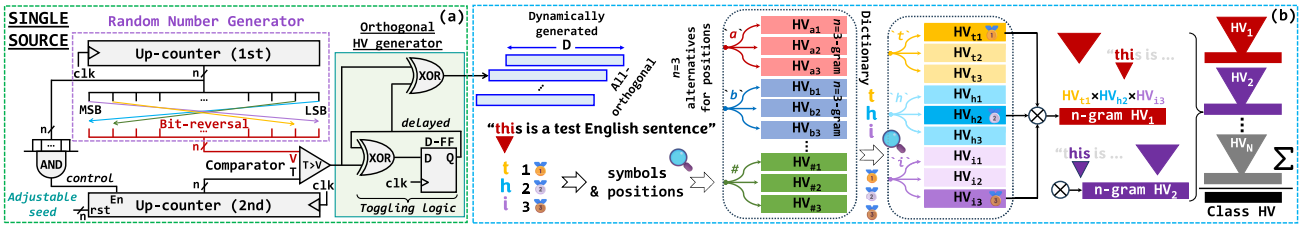

Fig. 5. Novel symbol vector generation. (a) Proposed generator architecture. (b) No-shift positional encoding with free vector generation.

The first design, **ID-VSA ❶**, addresses symbol-oriented vector generation. Orthogonal symbol HVs are produced using a single-source, lightweight, high-quality vector generator. Unlike baseline methods that rely on multiple random sources to provide near orthogonality between HVs, our method uses a single generator to produce orthogonal HVs for tasks such as pixel position encoding or character representation in language models. The second design, **ID-VSA ❷**, enhances input data. While few prior studies have investigated modifying input data for HDC [22], [23], our work shows how SC architectures can provide new perspectives on incoming data. Specifically, within **ID-VSA ❷**, we introduce the Gaussian Polygon, a novel method for processing image data, while retaining the high-quality symbol vectors from **ID-VSA ❶**. The third design, **ID-VSA ❸**, extends *orthogonal distribution strategies* for symbol-oriented applications. For cases requiring both symbols and level scalars, as in image processing, **ID-VSA ❹** extends the single-source vector generation concept to include not only symbols but also intensity vectors during polygon-based operations. Both **ID-VSA ❸** and **❹** benefit from the proposed QND. Finally, the fifth design, **ID-VSA ❺**, explores the use of single-source true-random inputs, which naturally provide orthogonal vectors for symbols. This approach is particularly promising for device-level implementations that can exploit intrinsic hardware noise, and for future analog-oriented designs.

A. *ID-VSA ❶: Single-Source Symbol Vector Generator*

In HDC, symbols rely on orthogonality (uncorrelation) between HVs to maintain accuracy and separability. To ensure this property, we propose a new encoding method that generates orthogonal symbol HVs using *van der Corput*(VDC) sequences. VDC sequences belong to the family of LD sequences, well studied in number and discrepancy theory. They have been recently proposed for SC designs due to their ability to uniformly distribute points across an interval [24]. In fact, a VDC sequence can serve as a simple RNG for high-quality SC bit-stream generation [25]. A VDC sequence is produced using a *radical inverse function* (RIF), which can be applied over any integer base $\mathcal{B} \geq 2$. The defining property of these sequences is their LD, ensuring that points are spread evenly throughout the unit space. This makes them particularly valuable in applications such as numerical integration and quasi-Monte Carlo simulations [26]. To generate a VDC sequence, natural numbers are mapped to the unit interval $[0, 1)$ using base-$\mathcal{B}$ expansions, referred to as VDC-$\mathcal{B}$. Each term is obtained by reversing the digits of a natural number expressed in base $\mathcal{B}$, then interpreting the reversed digits as a fractional value. This digit-reversal operation is essential, as it produces the uniform coverage that characterizes VDC sequences. For example, the decimal number $(45)_{10}$ can be expressed as $(231)_4$ in base-4. Reversing the digits gives $(0.132)_4$, which corresponds to: RIF $= 1 \times 4^{-1} + 3 \times 4^{-2} + 2 \times 4^{-3} = 1/4 + 3/16 + 2/64 = 30/64 = (0.46875)_{10}$.

ID-VSA ❶ leverages the VDC sequences, where we develop a lightweight architecture using a single VDC pattern to generate multiple independent HVs. The design is suitable for multi-symbol HDC systems. The proposed architecture relies on simple building blocks—XOR gates, up-counters, and D-flip flops (D-FFs)—to maintain orthogonality while using only a single random sequence generator. This eliminates the need for multiple generators, reducing both hardware cost and design complexity. As a result, the system is compact yet capable of preserving the high orthogonality required for accurate symbolic encoding. Fig. 5(a) shows the proposed architecture. HVs are generated using a VDC-2 sequence, created by an up-counter with a bit-reversal mechanism, which can be implemented efficiently in hardware by *hardwiring*. Since the system operates from a single VDC pattern, these hardwired connections remain fixed and incur no additional hardware cost. The resulting VDC-2 values (V) are compared dynamically against reference values (called T) from a second up-counter. This counter is controlled by the single-source

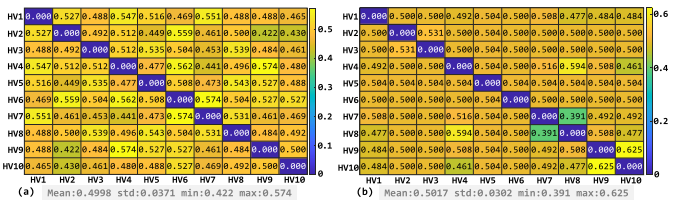

Fig. 6. An example of normalized Hamming distance heatmap plots demonstration for (a) baseline HDC/VSA employing randomly generated HVs based on software and (b) **ID-VSA** ❶ approach utilizing single-source HV generator design. 10 HVs are selected for each method.

sequence generator's enable signal. Each comparison outputs a "1" if $T > V$ and a "0" otherwise. The generated bit then passes through an orthogonal vector generation block, where it undergoes two successive XOR operations: one with the incoming bit and another with a delayed version of the previously generated bit. Assuming one bit of the HV is generated per cycle, a complete HV of length D is generated in D cycles. The next HV is then generated in the subsequent D cycles using the next value from the second up-counter.

The reproducibility and determinism properties of the proposed method ensure that, across different runs, each generated symbol vector maps to the same HV. This reproducibility stems from the deterministic structure of the random sources: we use an up-counter that traverses the same cycle at each run, and exploit quasi-random generation enabled by hardwired (and thus deterministic) bit-reversal connections. Consequently, symbol HV generation is identical across runs and during training and inference, provided the same setup (including the same V and T initialization) is used. For symbols that appear many times (e.g., *letters* in language text), the symbol set is limited and can optionally be cached and reused with only $D \times$ *symbol count* storage (e.g., $26 \times D$-bit for English). When the symbol space is excessive (e.g., when image pixel row/column positions map to a large scale), caching may become infeasible; this is where **ID-VSA** ❶ is beneficial. A large number of position symbols can be generated dynamically yet identically across runs (i.e., the same position index always yields the same HV by setting the adjustable seed in the second up-counter in Fig. 5). This facilitates large-scale HDC applications where dynamic symbol growth is critical.

1) Proof of Concept: To demonstrate the effectiveness of the proposed single-source vector generator, we evaluated the similarity among HVs produced by our design and compared them with HVs generated using conventional software-based methods. The similarity was measured using the normalized *Hamming distance*, defined as: $Hamming(HV1, HV2) = 1/D \sum_{i=1}^{D} (\overrightarrow{HV1_i} \oplus \overrightarrow{HV2_i})$, where $\overrightarrow{HV1}$ and $\overrightarrow{HV2}$ are two HVs of length D and $\oplus$ represents bit-wise XOR. A distance of 0.5 indicates full orthogonality, while a value of 0 implies maximum correlation between HVs. Fig. 6 shows the heatmap plots of the pairwise *Hamming distance* among 10 randomly selected HVs generated by the baseline approach [Fig. 6(a)] and by our method [Fig. 6(b)]. The results confirm that our generator consistently produces ideal orthogonal HVs, with most values clustering around 0.5. Although a few minor deviations are visible, the distribution clearly demonstrates that the proposed framework achieves the level of orthogonality required for reliable HDC operation.

2) Flexible Multi-assignment Symbol Encoding Without Shifting: Since HV generation in hardware is now straightforward and flexible, we assign multiple position-specific HVs to each symbol. For example, in English, 27 characters (including space) yield 81 HVs in a $n = 3$-gram scenario. This design simplifies n-gram processing while preserving positional information directly through vector assignment. While software-based vector generation may be adequate for edge deployments with fixed symbol sets, many real-world applications require a more dynamic, hardware-independent approach. In such cases, the number of symbols or the number of vectors assigned per symbol can change over time, making static methods impractical. To address this, we propose a new encoding method for n-gram text processing that avoids traditional shift-and-rotate operations. In our approach, each symbol is assigned n orthogonal HVs, one for each position in the n-gram window. For example, in a 3-gram ($n = 3$), every symbol has three associated HVs, corresponding to its appearance in the first, second, or third position in the n-gram window. Fig. 5(b) illustrates this no-shift, multi-assignment method. For instance, consider the first 3-gram in the text sequence "**thi**": the symbol "**t**" in the first position uses its first HV, "**h**" in the second position uses its second HV, and "**i**" in the third position uses its third HV. Then, these three HVs are bound (multiplied using XOR) together. As the sliding window moves forward (e.g., to form the next 3-gram "**his**"), the process repeats, assigning HVs according to symbol position. After generating HVs for all n-grams, they are accumulated bit-wise using a population count operation. Finally, all n-gram HVs belonging to the same class (e.g., text samples from the same language in a classification task [2]) are aggregated into a class HV for training and inference.

A central innovation of the **ID-VSA** ❶ design is the shift from a *Store-and-Retrieve* model to a *Compute-on-Demand* paradigm for symbol HVs. In traditional HDC implementations, HVs for all possible symbols (e.g., an entire alphabet or pixel position set of length M) are pre-computed and stored in a look-up table (LUT) in memory. This requires $M \times D$ bits of on-chip storage, which becomes a major area bottleneck as the symbol set size M or HV dimensionality D scales. **ID-VSA** ❶ eliminates this requirement by leveraging the deterministic properties of quasi-random sequences. Since the VDC-2 generator and its associated up-counters are initialized to a known reset state, each symbol index consistently maps to the same orthogonal HV. Consequently, the hardware does not need to store individual symbol HVs for the duration of a task. Instead, it regenerates the required HV on demand during encoding. This strategy allows the system to store only the final accumulated class HVs, leading to a significant reduction in area footprint and power consumption. For unstructured vocabularies (e.g., NLP tokens), symbols are linked to unique indices via enumeration by their ASCII code, enabling direct HV generation from the corresponding index without traversing priors, thus supporting $\mathcal{O}(1)$ access in D cycles.

B. *ID-VSA* ❷: *Data Pre-processing via Gaussian Polygon*

After generating HVs, we introduce a new pre-processing stage called Gaussian Polygon. Previous works have explored vector-based feature extraction, often by applying a histogram of oriented gradients (HOGs) for features [22], [23].

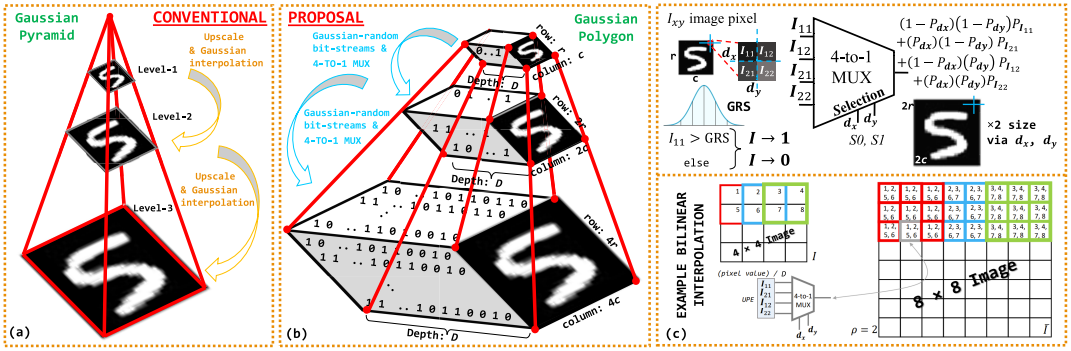

Fig. 7. From conventional Gaussian Pyramid to Gaussian Polygon: (a) Gaussian Pyramid in image processing, (b) Gaussian Polygon over bit-streams, and (c) novel method: 4-to-1 MUX architecture for upscaling sourced with Gaussian RNGs via sampling for Gaussian interpolations.

Our approach aims to further enhance the retraining strategy by incorporating robust pre-processing. We leverage SC to perform lightweight, efficient operations directly in the vector domain of input data. This keeps all computations within the bit-stream representation and extracts informative features for smooth training directly from the already generated HVs, thereby reducing computational overhead.

A well-known technique in image processing is the Gaussian Pyramid, which refines data representation and improves learning through upscale Gaussian interpolation [22], as illustrated in Fig. 7(a). In our framework, however, we avoid pixel-domain operations and keep all processing in the HV domain (i.e., the bit-stream domain enabled by SC). To achieve this, we propose an alternative method called Gaussian Polygon, shown in Fig. 7(b). Here, the image is treated as a 3-D structure consisting of rows (r), columns (c), and HV length (D). During upscaling and Gaussian-like interpolation, a polygonal form emerges naturally across these dimensions. Leveraging SC, we employ a 4-to-1 MUX sliding kernel to double the image size, applying linear interpolation along both the x (width) and y (height) directions on the xy- plane. Repeated interpolation across rows and columns produces an upscale representation similar to the Gaussian Pyramid, but fully realized in the vector/bit-stream domain.

Formally, let I be a 2-D matrix with coordinates (x, y). Consider a rectangular block within I bounded by four corner values: (x_1, y_1), (x_2, y_1), (x_1, y_2), and (x_2, y_2). To estimate a new value $I(x, y)$ inside this region, bilinear interpolation is applied on the known values at the four corner coordinates: $I(x, y) \approx a_{11}I(x_1, y_1) + a_{21}I(x_2, y_1) + a_{12}I(x_1, y_2) + a_{22}I(x_2, y_2)$, where the coefficients are defined as

$$a_{11} = \left[(x_2 - x)\,(y_2 - y)\right] / \left[(x_2 - x_1)\,(y_2 - y_1)\right]$$
$$a_{21} = \left[(x - x_1)\,(y_2 - y)\right] / \left[(x_2 - x_1)\,(y_2 - y_1)\right]$$
$$a_{12} = \left[(x_2 - x)\,(y - y_1)\right] / \left[(x_2 - x_1)\,(y_2 - y_1)\right]$$
$$a_{22} = \left[(x - x_1)\,(y - y_1)\right] / \left[(x_2 - x_1)\,(y_2 - y_1)\right] \quad [27],\ [28].$$

By normalizing the values in matrix I, the rectangular region is mapped into a unit square with corners at $(0, 0)$, $(1, 0)$, $(0, 1)$, and $(1, 1)$ [29]. This yields the simplified form

$$I(x, y) \approx (1 - x)\,(1 - y)\,I(x_1, y_1) + (x)\,(1 - y)\,I(x_2, y_1) + (1 - x)\,(y)\,I(x_1, y_2) + (x)\,(y)\,I(x_2, y_2). \quad (1)$$

For an input image $I_{(r \times c)}$, applying this filter produces a new version $\bar{I}_{(\varrho r \times \varrho c)}$ with scaling factor ϱ. The new image, $\bar{I}_{(\varrho r \times \varrho c)}$, has a size increased by ϱ compared to I. Equation (1) can then be rewritten as: $I(x, y) = (1 - dx)(1 - dy)I_{11} + (1 - dx)(dy)I_{12} + (dx)(1 - dy)I_{21} + (dx)(dy)I_{22}$, where I_{11}, I_{12}, I_{21}, I_{22} are neighboring pixels, and dx, dy are the relative positions of the new pixel $I(x, y)$ with respect to vertices, expressed in multiples of $1/\varrho$: i.e., $m \times 1/\varrho$, where $m \in \{1, \ldots, \varrho\}$.

In our SC-based formulation, a Gaussian random source (GRS) is used to convert each pixel intensity into a bit-stream. The GRS samples values from a Gaussian distribution, which are then applied during vector generation. A single 1-D sequence of sampled values is reused for all pixel intensities, ensuring the required correlation for proper MUX operation. To encode a pixel with intensity I, only D comparisons against the sampled values are needed, making the approach hardware-efficient. Relative positions dx and dy are also encoded in bit-streams. The probabilities for the neighboring pixels and their relative positions are denoted as $P_{I_{11}}$, $P_{I_{12}}$, $P_{I_{21}}$, $P_{I_{22}}$, P_{dx}, and P_{dy}. The stochastic interpolation for the pixel value at position (x, y) becomes: $P_{I(x,y)} = (1 - P_{dx})(1 - P_{dy})P_{I_{11}} + (1 - P_{dx})(P_{dy})P_{I_{12}} + (P_{dx})(1 - P_{dy})P_{I_{21}} + (P_{dx})(P_{dy})P_{I_{22}}$. This formulation behaves like a 4-to-1 MUX in the stochastic domain, extending the 2-to-1 MUX concept discussed earlier in Section II-A. In this setup, dx and dy act as stochastic selectors for the MUX, as illustrated in Fig. 7(c). This analogy is leveraged to design a novel SC-based upscaling filter. The hardware-friendly nature of SC enables an efficient implementation of interpolation. Fig. 7(c) shows an example application of the proposed stochastic filter on a 4×4 image enlarged by a scaling factor of $\varrho = 2$. Each new pixel is estimated by the stochastic interpolation formula, where dx and dy determine the interpolation weights for the neighboring pixels.

The proposed Gaussian Polygon method processes intensity HVs (I), followed by the standard steps of record-based encoding illustrated in Fig. 4. Meanwhile, symbol HVs for pixel positions are produced by our single-source orthogonal vector generator in **ID-VSA** ❶, which can generate additional HVs on demand, even after several iterations. These two sets of vectors are then bound together with XOR operations during encoding (Step ❸ in Fig. 4) and accumulated into the class HVs (Step ❹ in Fig. 4). In contrast, conventional HDC often requires retraining over multiple epochs: all samples from the same class are read during the first epoch to construct the corresponding class HV, and in the second epoch, a validation step checks whether each incoming sample is correctly classified [30]. For correctly classified samples,

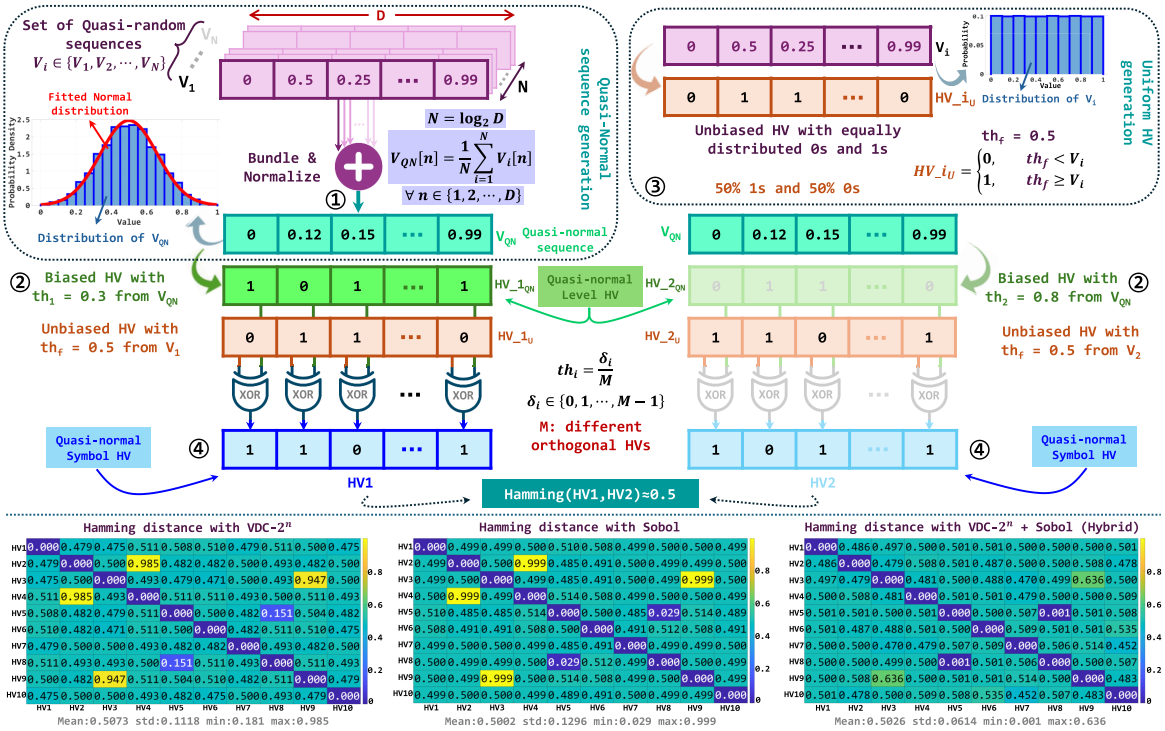

Fig. 8. General overview of constructing orthogonal HVs from *quasi-normal distribution* inspired by the central limit theorem.

their contribution is highlighted in the current class vector at a rate proportional to the learning rate (η); otherwise, their contribution is subtracted at the same rate (see Algorithm 1).

While retraining across multiple epochs (*three* for the baseline in this work) can enhance learning performance, it adds computational complexity due to iterative, η-dependent updates and similarity-based calculations over class HVs, making it less suitable for edge deployment. In contrast, our proposed multiscale training strategy using the `Gaussian Polygon` achieves comparable accuracy with significantly reduced complexity. By employing only a lightweight `MUX` operation combined with Gaussian-sampled data, our method efficiently enriches features without retraining. During training, each input image is processed at three scales: the original resolution (e.g., 28×28 for MNIST), a first-level upscaled version (56×56), and a second-level upscaled version (112×112). Each scaled version is independently encoded and contributes to the corresponding class HV, effectively repeating the same encoding step across multiple resolutions of the same sample. This multiscale training approach eliminates the need for η tuning or similarity-based updates and results in a compact, hardware-friendly method. By relying only on lightweight `MUX` operations and Gaussian-based randomness, the method delivers accuracy comparable to retraining-based HDC while providing a compact, edge-friendly solution.

C. *ID-VSA ❸ and ❹: Single-Source Symbol and Level Vector Generation via QND*

While the LD-uniform distribution of VDC sequences is highly effective for generating orthogonal symbol HVs (as detailed in Section III-A, **`ID-VSA ❶`**), we now ask: *Can a single source also provide Gaussian-dependent vectors required for* `Gaussian Polygon` *operations in* **`ID-VSA ❷`**?

To answer this, we introduce a method that transforms uniformly distributed LD sequences into a **QND** leveraging the principles of the central limit theorem (CLT). According to the CLT, the sum of a large number of independent and identically distributed (i.i.d.) random variables approaches a normal distribution, regardless of the underlying distribution of the individual variables. Inspired by this, we generate a new uniformly distributed sequence by aggregating and normalizing multiple quasi-random LD sequences (e.g., Sobol and powers-of-2 bases of VDC sequences, denoted as VDC-2^n). This aggregated sequence serves as the foundation for building distinct and orthogonal HVs.

Let V_i represent a quasi-random sequence of D points. Each sequence is a set of quasi-random variables, $V_i = \{V_i[n] \;\; \forall\; n = 1, 2, \ldots, D\}$, where each $V_i[n]$ is drawn from a uniform distribution over the interval $[0, 1)$. This is denoted as $V_i \sim U(0, 1)$ with a mean $\mu \approx 0.5$ and variance of $\sigma^2 \approx 1/12$. To generate the quasi-normal sequence V_{QN}, we normalize the element-wise sum of N distinct V_i sequences as

$$V_{QN}[n] = \frac{1}{N}\sum_{i=1}^{N} V_i[n], \; n = 1, \ldots, D \tag{2}$$

where $N = \log_2 D$. Empirical results indicate that, for sufficiently large N, V_{QN} approximates a normal distribution. *Goodness-of-fit* tests, such as the Jarque–Bera test [31], confirm this, often failing to reject normality even for moderate D. This indicates that structural dependencies in the aggregated sequence do not introduce significant skewness or kurtosis.

Once the quasi-normal sequence is obtained, we generate nearly orthogonal HVs through the steps illustrated in Fig. 8: The process begins by element-wise bundling (aggregating) a set of N quasi-random sequences, followed by normalization to the $[0, 1]$ interval (Fig. 8 ①). Next, we generate a set of

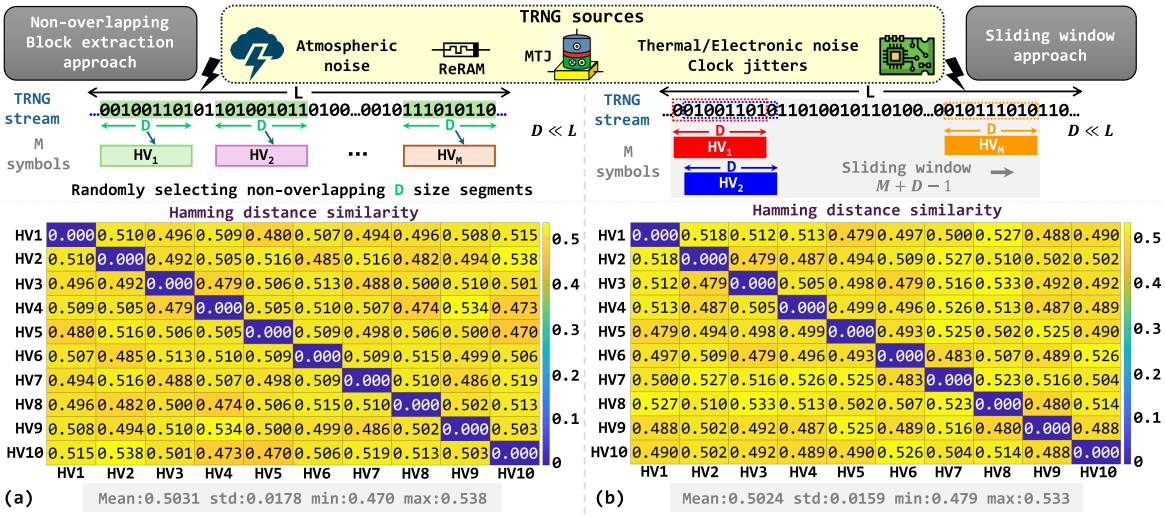

Fig. 9. Obtaining HVs from a TRNG stream. (a) Non-overlapping block extraction approach. (b) Sliding window approach.

M biased HVs (HV_i_{QN}) according to dynamically adjusted thresholds defined as $th_i = \delta_i/M, \delta_i \in \{0, 1, \ldots, M-1\}$ (Fig. 8 ②). The next step involves generating uniformly distributed HVs (HV_i_U) from quasi-random V_i sequences with a fixed threshold of $th_f = 0.5$ (Fig. 8 ③). Finally, we perform element-wise XOR operations on HV_i_{QN} and HV_i_U ($HV\_i_{QN} \oplus HV\_i_U$) to produce the final orthogonal HV (Fig. 8 ④). A key advantage of this method is that HV generation can be parallelized by adopting multiple th_i concurrently. Another benefit is that a single-source data stream, rather than multiple D-dimensional random inputs, remains sufficient to produce independent and adaptive HDC representations. Since LD sequences are deterministic and stable, their aggregation (bundling) and normalization can be performed offline, ensuring that accumulated values remain consistent across deployment.

We further evaluate the orthogonality of the generated HVs using quasi-normal sequences derived from VDC-2^n, Sobol, or a hybrid of both. The Hamming distance results for 10 randomly selected HVs are shown at the bottom of Fig. 8. In the hybrid approach, some symbol HVs are generated from Sobol-based aggregated–normalized quasi-normal sequences, while others are derived from VDC-2^n-based counterparts. Orthogonality improves when combining the two quasi-random sources: the measured distances cluster much closer to 0.5 with lower variance, compared to the higher fluctuations observed when using only Sobol or only VDC-2^n sequences. Up to this point, we have focused only on symbol HVs, as architecture we refer to as **ID-VSA ❸**. This design is suited for symbol-only applications such as language processing, where high-quality symbol generation is the primary requirement. However, our broader objective is also to leverage the level HVs, particularly because normal (Gaussian)-like distributions are advantageous for `Gaussian Polygon` operations.

In **ID-VSA ❹**, we extend the design so that the same quasi-normal sequence is used not only for symbol generation but also for producing level HVs required in `Gaussian Polygon` upscaling. This unified approach benefits both symbol and level vector generation, while also supporting data processing within the `Gaussian Polygon` during upscaling operations enabled by SC-`MUX` processing. Fig. 8 illustrates how the threshold-based level vectors are generated by comparing the quasi-normal sequence directly against the intensity thresholds. By varying these thresholds, we generate HVs for all required intensity values, which can then be supplied to the SC-`MUX` for upscaling.

D. *ID-VSA ❺: True-Randomness for Symbols*

We further evaluate the functionality of our framework by utilizing TRNG streams, targeting device-level randomness. This design looks ahead to future implementations that exploit the inherent random behavior of physical devices, such as thermal or electrical *noise*. The key advantage of this approach is that it does not require dedicated hardware for symbol HV generation. TRNG binary streams are "*true*" random; meaning that each bit has a 0.5 probability of being logic-1. Given a sufficiently long stream of length L, we can randomly select any subset of length D ($D \ll L$), and utilize it as a unique HV, passing it directly to the encoding stage for further processing. To obtain M symbol HVs, this process can be repeated by extracting M *nonoverlapping blocks* from the TRNG stream, as illustrated in Fig. 9(a). Alternatively, M near-orthogonal symbol HVs can be generated using a *sliding window* approach over a TRNG stream of length $M + D - 1$. In this approach, the first HV is formed by selecting an initial D window, and each subsequent HV_i is obtained by shifting this window by i positions, as shown in Fig. 9(b). The effectiveness of both methods is validated through Hamming distance analysis, which confirms that HVs derived from TRNG streams preserve near-orthogonality and are well-suited for symbol vector generation. We further distinguish this stream-extraction sliding-window mechanism from permutation/shift baselines that shift an existing seed HV; **ID-VSA ❺** instead forms symbol HVs directly from the TRNG stream ($M + D - 1$), thereby mitigating correlation and improving robustness in multiscale settings. In summary, **ID-VSA ❺** employs single-source, true-random symbol vectors. For image processing applications, it integrates seamlessly `Gaussian Polygon` operations introduced in **ID-VSA ❷**, enabling consistent processing across both symbol- and image-based tasks. While **ID-VSA ❺** offers the simplest and lowest-overhead path for symbol HV generation by partitioning

long TRNG streams into HVs, it is inherently limited to producing uniform distributions. In contrast, the deterministic, logic-based methods (**ID−VSA** ❸,❹) enable the generation of QNDs, which are essential for multiscale Gaussian Polygon operations. The Hamming-distance heatmaps shown in Figs. 6, 8, and 9 are intended as illustrative sanity checks rather than stand-alone quantitative comparisons. For finite-dimensional HVs, pairwise hamming distances naturally exhibit statistical variability, and isolated deviations from 0.5 are expected even for statistically independent vectors. Accordingly, the orthogonality quality is more reliably characterized by the distribution statistics (specifically the mean and variance) of the off-diagonal Hamming distances, rather than by individual entries or visual inspection alone. For completeness, the mean, standard deviation (std), and minimum/maximum values across all off-diagonal pairs are reported alongside each heatmap.

IV. Design Evaluation

A. Experimental Setup

This section evaluates the proposed designs in terms of *classification accuracy* and *hardware efficiency*. Software-based experiments were conducted on a system equipped with an Intel Core i7 CPU (2.30 GHz), 16 GB of RAM, and an NVIDIA RTX-3050 GPU with 4 GB of memory. Hardware designs were implemented in Verilog and synthesized using Synopsys Design Compiler v2024.09-SP4 with a 45 nm cell library with the target operating frequency of 1 GHz.

B. Performance Measurements

We evaluate the accuracy of the proposed **ID−VSA** framework across both text- and image-based benchmarks. Table II reports the results for an n-gram-based classification task on the 21-class European language dataset [40]. For the TRNG-based method, we employed two true-random sources: [33] (TRNG1) and [34] (TRNG2). Both sources have successfully passed all NIST statistical tests for randomness. For comparison, we also report baseline architecture results from the literature: 1) the Sobol method with optimized thresholds [32]; 2) a software-based random approach using MATLAB's `rand()` function [2]; 3) an LFSR-based pseudo-random method [32], which requires carefully chosen seeds and polynomial to maintain orthogonality; and 4) TRNG sources using sliding window approach. Each baseline is shown in its best-performing configuration, including optimized thresholds (the unbiased point) during HV generation and the best n-gram window sizes (n). Although our proposed method achieves its highest accuracy with $n = 3$, which is lower than the optimal n values for some baselines, it still provides comparable or better accuracy overall. These results demonstrate the strong orthogonality, efficiency, and effectiveness of the proposed encoding scheme. The results show that the proposed **ID−VSA** ❶ performs well for the shorter vector lengths (D = 128, 256), making them well suited for resource-constrained edge systems. **ID−VSA** ❺ (TRNG-based) also demonstrates strong results at lower dimensionalities. For **ID−VSA** ❸, the hybrid approach for the symbol vector generation gives better results compared to the sole LD sequence-based vector generation.

Table III reports classification accuracy and $F1$-scores on three image datasets: MNIST [41], Pneumonia MNIST (P.MNIST) [42], and breast MNIST (B.MNIST) [43]. The baselines include four random sources for position vectors:

TABLE II

Comparison of No-Shift Positional Encoding With Free Vector Generation in Language Classification (Symbol-Only)

Encoding Approach	D	n-gram	Threshold	Acc.(%)
Baseline-Optimal Sobol [32]	128	4	0.74	60.74
Baseline-Software-random [2]		4	0.5	52.04
Baseline-LFSR w/ random seed [32]		4	0.5	51.16
Baseline-T [2]-sliding window, ✣		3	No tuning	73.04
Baseline-T [2]-sliding window, ★		3	No tuning	69.16
ID−VSA ❶, VDC-2		3	No tuning	79.10
ID−VSA ❸, *QND* (Sobol)		3	No tuning	52.00
ID−VSA ❸, *QND* (VDC)		3	No tuning	52.56
ID−VSA ❸, *QND* (Hybrid)		3	No tuning	63.71
ID−VSA ❺-block extraction, ✣		3	No tuning	74.24
ID−VSA ❺-block extraction, ★		3	No tuning	75.12
ID−VSA ❺-sliding window, ✣		3	No tuning	73.80
ID−VSA ❺-sliding window, ★		3	No tuning	75.31
Baseline-Optimal Sobol [32]	256	4	0.70	79.03
Baseline-Software-random [2]		4	0.5	69.24
Baseline-LFSR w/ random seed [32]		4	0.5	68.61
Baseline-T [2]-sliding window, ✣		3	No tuning	84.51
Baseline-T [2]-sliding window, ★		3	No tuning	83.28
ID−VSA ❶, VDC-2		3	No tuning	87.40
ID−VSA ❸, *QND* (Sobol)		3	No tuning	65.82
ID−VSA ❸, *QND* (VDC)		3	No tuning	54.75
ID−VSA ❸, *QND* (Hybrid)		3	No tuning	82.09
ID−VSA ❺-block extraction, ✣		3	No tuning	85.56
ID−VSA ❺-block extraction, ★		3	No tuning	86.14
ID−VSA ❺-sliding window, ✣		3	No tuning	86.05
ID−VSA ❺-sliding window, ★		3	No tuning	85.67
Baseline-Optimal Sobol [32]	512	4	0.70	89.47
Baseline-Software-random [2]		4	0.5	83.03
Baseline-LFSR w/ random seed [32]		4	0.5	82.22
Baseline-T [2]-sliding window, ✣		3	No tuning	89.83
Baseline-T [2]-sliding window, ★		3	No tuning	89.68
ID−VSA ❶, VDC-2		3	No tuning	89.14
ID−VSA ❸, *QND* (Sobol)		3	No tuning	55.43
ID−VSA ❸, *QND* (VDC)		3	No tuning	64.96
ID−VSA ❸, *QND* (Hybrid)		3	No tuning	83.31
ID−VSA ❺-block extraction, ✣		3	No tuning	92.17
ID−VSA ❺-block extraction, ★		3	No tuning	92.18
ID−VSA ❺-sliding window, ✣		3	No tuning	91.79
ID−VSA ❺-sliding window, ★		3	No tuning	91.98
Baseline-Optimal Sobol [32]	1024	4	0.70	93.78
Baseline-Software-random [2]		4	0.5	91.15
Baseline-LFSR w/ random seed [32]		4	0.5	90.56
Baseline-T [2]-sliding window, ✣		3	No tuning	93.03
Baseline-T [2]-sliding window, ★		3	No tuning	92.88
ID−VSA ❶, VDC-2		3	No tuning	91.01
ID−VSA ❸, *QND* (Sobol)		3	No tuning	80.06
ID−VSA ❸, *QND* (VDC)		3	No tuning	82.80
ID−VSA ❸, *QND* (Hybrid)		3	No tuning	93.20
ID−VSA ❺-block extraction, ✣		3	No tuning	95.19
ID−VSA ❺-block extraction, ★		3	No tuning	95.20
ID−VSA ❺-sliding window, ✣		3	No tuning	94.93
ID−VSA ❺-sliding window, ★		3	No tuning	95.00

✣→TRNG1 [33], ★→TRNG2 [34]

TABLE III

Comparison of Image Classification Results (Symbol and Level Vectors)

Encoding Approach (D=1024)	MNIST		P.MNIST		B.MNIST	
	Acc.(%)	F1	Acc.(%)	F1	Acc.(%)	F1
Baseline-P w/o retrain [35]	81.12	0.79	90.23	0.89	69.19	0.69
Baseline-P w/ retrain [36]	84.06	0.83	92.81	0.91	71.02	0.71
Baseline-S w/o retrain [37]	81.12	0.79	88.88	0.87	71.02	0.70
Baseline-S w/ retrain [32]	86.02	0.85	92.36	0.91	74.46	0.73
Baseline-H w/o retrain [38]	81.02	0.79	89.26	0.88	68.78	0.68
Baseline-H w/ retrain [38]	85.12	0.84	89.73	0.88	78.34	0.77
Baseline-V w/o retrain [39]	81.02	0.79	89.21	0.88	69.36	0.69
Baseline-V w/ retrain [39]	89.39	0.88	92.02	0.91	75.13	0.74
Baseline-T w/o retrain-sliding window,✣	79.52	0.80	79.92	0.80	66.30	0.66
Baseline-T w/ retrain-sliding window,✣	83.44	0.83	90.99	0.91	70.17	0.70
Baseline-T w/o retrain-sliding window,★	79.58	0.80	79.73	0.80	67.40	0.67
Baseline-T w/ retrain-sliding window,★	85.44	0.85	90.22	0.90	71.27	0.71
ID−VSA ❷ 28 × 28	82.02	0.87	88.12	0.87	69.36	0.69
ID−VSA ❷ 56 × 56	89.14	0.88	90.27	0.89	69.41	0.69
ID−VSA ❷ 112 × 112	90.23	0.89	92.11	0.91	77.18	0.76
ID−VSA ❹ 28 × 28	86.19	0.86	84.26	0.84	71.12	0.71
ID−VSA ❹ 56 × 56	87.21	0.87	92.37	0.92	73.46	0.73
ID−VSA ❹ 112 × 112	89.32	0.89	92.41	0.92	72.54	0.73
ID−VSA ❺✣ 28 × 28	88.11	0.88	93.25	0.93	71.34	0.71
ID−VSA ❺✣ 56 × 56	89.23	0.89	93.42	0.93	71.46	0.71
ID−VSA ❺✣ 112 × 112	90.29	0.89	94.28	0.94	75.18	0.75
ID−VSA ❺★ 28 × 28	88.37	0.88	94.15	0.94	63.22	0.63
ID−VSA ❺★ 56 × 56	89.42	0.89	94.26	0.94	67.38	0.67
ID−VSA ❺★ 112 × 112	90.46	0.90	95.31	0.95	73.41	0.73
ID−VSA ❺-sliding window,✣ 28 × 28	87.33	0.87	89.38	0.89	69.18	0.69
ID−VSA ❺-sliding window,✣ 56 × 56	87.91	0.88	90.67	0.91	70.32	0.70
ID−VSA ❺-sliding window,✣ 112 × 112	88.01	0.88	91.18	0.91	71.67	0.71
ID−VSA ❺-sliding window,★ 28 × 28	88.18	0.88	91.06	0.91	69.06	0.69
ID−VSA ❺-sliding window,★ 56 × 56	88.31	0.88	91.18	0.91	69.36	0.69
ID−VSA ❺-sliding window,★ 112 × 112	89.61	0.89	91.35	0.91	71.48	0.71

✣→TRNG1 [33], ★→TRNG2 [34]

TABLE IV
CLASSIFICATION RESULTS ON UCIHAR [44] AND ISOLET [45]

Encoding Approach (D=1024)	UCIHAR Acc.(%)	UCIHAR F1	ISOLET Acc.(%)	ISOLET F1
Baseline-P w/o retrain [35]	86.14	0.86	86.08	0.85
Baseline-P w/ retrain [36]	93.27	0.93	92.16	0.92
Baseline-S w/o retrain [37]	88.09	0.88	86.22	0.86
Baseline-S w/ retrain [32]	93.31	0.93	92.41	0.92
Baseline-H w/o retrain [38]	86.23	0.86	86.37	0.86
Baseline-H w/ retrain [38]	93.18	0.93	93.42	0.93
Baseline-V w/o retrain [39]	86.07	0.86	86.14	0.86
Baseline-V w/ retrain [39]	96.21	0.96	93.38	0.93
Baseline-T w/o retraining-sliding window,✣	82.59	0.83	73.40	0.73
Baseline-T w/ retraining-sliding window,✣	91.18	0.91	83.13	0.83
Baseline-T w/o retraining-sliding window,★	82.59	0.83	73.01	0.73
Baseline-T w/ retraining-sliding window,★	90.77	0.91	83.46	0.83
ID-VSA ❶	96.34	0.96	94.26	0.93
ID-VSA ❸ *QND* (Sobol)	97.14	0.97	92.24	0.92
ID-VSA ❸ *QND* (VDC-2^n)	96.18	0.96	94.12	0.93
ID-VSA ❸ *QND* (Hybrid)	93.22	0.92	84.37	0.84
ID-VSA ❺✣	96.27	0.96	93.41	0.93
ID-VSA ❺★	96.08	0.96	90.36	0.90
ID-VSA ❺-sliding window✣	96.06	0.96	93.28	0.93
ID-VSA ❺-sliding window★	95.91	0.96	92.70	0.93

✣→TRNG1 [33], ★→TRNG2 [34]

TABLE V
HARDWARE COSTS ASSOCIATED WITH **HV Generation** USING THE BASELINE [2] AND **ID-VSA ❶, ❸, ❹** APPROACHES

Encoding Approach	HV length (D)	CPL* (ns)	Area (μm^2)	Power (mW)	Energy (nJ)	Area×Delay ($\mu m^2 \times ns$)
Baseline-full dictionary	256	0.33	5870	21.99	0.007	1937.1
Baseline-sliding window		0.10	2257	20.75	0.002	225.7
ID-VSA ❶		0.42	456	1.10	0.12	191
ID-VSA ❸		0.39	156	0.211	0.02	60.8
ID-VSA ❹ (level HV)		0.33	83	0.148	0.01	27.4
Baseline-full dictionary	512	0.33	10601	43.99	0.01	3498.3
Baseline-sliding window		0.10	4300	39.52	0.004	430.0
ID-VSA ❶		0.44	520	1.17	0.26	229
ID-VSA ❸		0.42	178	0.196	0.04	74.8
ID-VSA ❹ (level HV)		0.33	90	0.158	0.03	29.7
Baseline-full dictionary	1024	0.35	21190	82.91	0.03	7416.5
Baseline-sliding window		0.10	8384	77.06	0.008	838.4
ID-VSA ❶		0.43	575	1.31	0.58	247
ID-VSA ❸		0.39	193	0.224	0.09	75.3
ID-VSA ❹ (level HV)		0.33	98	0.169	0.06	32.3

*Critical Path Latency. Energy is reported per complete HV generation: $E = P \times \text{CPL} \times N_{\text{cycles}}$, where $N_{\text{cycles}} = D$ for **ID-VSA ❶,❸,❹** (bit-serial generation) and $N_{\text{cycles}} = 1$ for the baseline (memory fetch). The Baseline area values reported in this table assume a fully stored on-chip dictionary and sliding window requiring $M \times D$ and $M + D - 1$ bits respectively for M=26 symbols of dimensionality D={256, 512, 1024}. For small symbol sets, alternative implementations such as (i) TRNG-based sliding window extraction requiring only $M + D - 1$ bits, or (ii) cached dictionary storage for a limited alphabet ($M_{cached} \times D$), can significantly reduce memory requirements.

pseudo-random (Baseline-P), Sobol (Baseline-S), Hadamard (Baseline-H), and VDC (Baseline-V), each tested with/without retraining. Level vectors use the regular LFSR random sources in the baselines. Table III also includes sliding-window TRNG-based (Baseline-T) variants *with* and *without* retraining. For the proposed **ID-VSA** framework, results are reported at three `Gaussian Polygon` resolutions (28×28, 56×56, 112×112) with different **ID-VSA** alternatives.

On the MNIST dataset, retraining improves all baselines, with Baseline-V reaching 89.39% accuracy. Among the proposed methods, **ID-VSA ❺** achieves one of the highest scores, reaching 90.46% at 112×112. On P.MNIST, retrained baselines achieve up to 92.81%, whereas **ID-VSA ❺** TRNG2 at 112×112 attains the best overall performance with 95.31% accuracy and an $F1$ score of 0.95. **ID-VSA ❷** and **ID-VSA ❹** reach up to 92% accuracy when upscaling images to 112×112. On the more challenging B.MNIST dataset, retrained Baseline-H achieves 78.34%, the strongest among baselines. **ID-VSA ❷** achieves 77.18%, outperforming the TRNG-based design and approaching the Baseline-H retrained score. The sliding-window variants consistently demonstrate stable performance across resolutions, particularly when combined with TRNG-based encodings. Overall, increasing `Gaussian Polygon` resolution (up to 112×112) generally improves accuracy, except for the **ID-VSA ❹** in the B.MNIST dataset.

As observed in Table III, further interpolation in image-based tasks does not provide significant accuracy improvements. On MNIST, accuracy improves from 82.02% at 28×28 to 90.23% at 112×112, but saturates beyond this resolution. On P.MNIST, accuracy at 112×112 reaches 95.31%, already surpassing all baseline methods. For B.MNIST, the trend is even clearer: accuracy improves only marginally with resolution, peaking at 77.18% with **ID-VSA ❷** 112×112, which is comparable to the best baseline of 78.34%.

Table IV reports classification accuracy and $F1$-scores on two non-image datasets (UCIHAR [44] and ISOLET [45]) under different encoding methods. As these datasets are not image-based, `Gaussian Polygon` interpolation is not applied, similar to the language classification case. On UCIHAR, baselines perform comparably without retraining (around 82.59%–88.09%) and improve after retraining: Baseline-P, Baseline-S, and Baseline-H; each exceeds 93%, while Baseline-V achieves the highest baseline accuracy with 96.21%. On ISOLET, baseline accuracies range from 73.01%–86.37% without retraining and improve to 83.13%–93.42% with retraining, where Baseline-H and Baseline-V perform best. The proposed **ID-VSA** consistently matches or outperforms the baselines. On UCIHAR, **ID-VSA ❶**, **ID-VSA ❸** (Sobol), **ID-VSA ❸** (VDC-2^n), and **ID-VSA ❺** each achieve beyond 95% accuracy. On ISOLET, **ID-VSA** variants reach up to 94.26% accuracy with **ID-VSA ❶** surpassing all baselines.

C. Hardware Efficiency

We evaluate the hardware efficiency of the proposed framework against the baseline HDC architecture with retraining over *three* epochs. We synthesized the proposed HV generator designs, **ID-VSA ❶** and **ID-VSA ❸** (shown in Figs. 5 and 8) and compared against baseline implementations [2], [46]. The synthesis results are summarized in Table V. The large area gap observed in Table V primarily stems from a fundamental architectural choice: dictionary-based memory storage versus on-chip logic generation. The baseline design stores and accesses memory-resident symbol and position HV dictionaries along with the associated read and control logic, which dominates the area footprint. For example, at D = 1024, the baseline occupies 21,190 μm^2, whereas the **ID-VSA** variants require 98–575 μm^2. In contrast, **ID-VSA** replaces large HV tables with compact on-chip generation logic and minimal persistent state (counters, `XOR` gates, and lightweight control). In addition, the proposed generators operate in a bit-serial manner: generating a complete D-bit HV requires $N_{\text{cycles}} = D$ cycles (one bit per cycle), whereas the baseline retrieves a pre-stored HV in a single cycle ($N_{\text{cycles}} = 1$). Accordingly, the energy values in Table V are reported per complete HV generation and naturally account for the D-cycle generation cost in our designs. For instance, at D = 1024, the baseline consumes 0.03 nJ per retrieval, compared to 0.06–0.58 nJ for bit-serial generation in the **ID-VSA** variants. The baseline architecture must store a full HV dictionary whose size scales with both D and the number of symbols or positions (M), i.e., on the order of $O(M_{\text{dictionary}} \times D)$ bits. This memory requirement directly drives the area growth. As a representative example, for position HVs, increasing D from

TABLE VI

MEMORY FOOTPRINT COMPARISON FOR HDC BASELINES WITH FULL-DICTIONARY AND SLIDING WINDOW EXTRACTION (M = 26)

HV length (D)	Baseline-T Full dictionary($M\times D$ bits)	Baseline-T Sliding Window($M+D-1$ bits)	Memory reduction (%) ↓
256	6,656	281	95.78
512	13,312	537	95.96
1024	26,624	1049	96.06

TABLE VII

HARDWARE COSTS ASSOCIATED WITH **Retraining** PROCESS USING THE BASELINE [46] AND **ID-VSA** ❷ APPROACHES

Encoding Approach	HV length (D)	CPL* (ns)	Area (μm^2)	Power (mW)	Energy (pJ)	Area×Delay ($\mu m^2 \times ns$)
Baseline	256	0.36	8439	43.43	15.63	3038
ID-VSA ❷		0.56	694	1.59	227.94	388
Baseline	512	0.38	16859	82.70	31.43	6406
ID-VSA ❷		0.50	787	1.89	483.84	393
Baseline	1024	0.43	33787	144.71	62.22	14528
ID-VSA ❷		0.50	864	1.99	1018.88	432

*Critical Path Latency. Energy is reported per complete retraining update: $E = P\times\text{CPL}\times N_{\text{cycles}}$, where N_{cycles}=1 for the baseline retraining block (bit-parallel similarity/update) and $N_{\text{cycles}} = D$ for **ID-VSA** ❷ (bit-serial operation). For multi-epoch retraining, the total training energy scales linearly with the number of epochs.

256 to 1024 increases the baseline (full dictionary) area from 5,870 to 21,190μm^2, while the corresponding area of **ID-VSA** ❸ grows only modestly, from 156 to 193 μm^2.

For the **ID-VSA** ❸ design, we excluded the cost of generating the quasi-normal sequence, as it can be prepared offline in software. So, we read the statically generated quasi-normal sequence from memory and use it to generate the quasi-normal HV. By contrast, the **ID-VSA** ❶ design incurs additional hardware cost due to the more complex structure required to generate orthogonal HVs (see Fig. 5). We also report the hardware cost for generating the level HVs using the quasi-normal approach (**ID-VSA** ❹). This cost includes the element-wise comparison between the quasi-normal sequence and the dynamically adjusted threshold of intensities, th_i (see Section III-C). As shown in Table V, this comparison-based level-HV generation method remains particularly lightweight (e.g., 83/90/98 μm^2 for D = 256/512/1024), as it primarily consists of a simple comparator and minimal control logic rather than a full orthogonal HV generation pipeline. Consequently, its power and energy consumption are also low (e.g., 0.148–0.169 mW and 0.01–0.06 nJ). For **ID-VSA** ❺, hardware costs associated with TRNG generation are not included in Table V, as TRNG implementation is technology-dependent and orthogonal to the HV-generation micro-architecture evaluated in this work. TRNG streams can be realized using different device technologies such as CMOS [47], resistive RAM (ReRAM) [48], or magnetic tunnel junctions (MTJs) [49]. Prior work demonstrated TRNG implementations with very low per-bit cost; for example, [50] reports a TRNG design with an area of 2 μm^2 and an energy consumption of 20 fJ/bit. In this study, **ID-VSA** ❺ assumes to receive TRNG streams from an external input. For workloads with small symbol alphabets (e.g., M = 26 for English letters), baseline designs can reduce memory by avoiding full dictionary storage. TRNG-based sliding-window extraction can reduce persistent memory from $M \times D$ bits to $M + D - 1$ bits, approximately a 96% reduction for D = 1024 (Table VI). Alternatively, caching only frequently used symbols further reduces storage to $M_{cached} \times D$. However, these optimizations primarily benefit small and static symbol sets. For large positional encodings (e.g., image row–column indices) or dynamically growing symbol dictionaries, M becomes large, and memory savings diminish. In contrast, **ID-VSA** eliminates explicit dictionary storage and enables on-the-fly generation of HVs with controllable statistical properties. This allows **ID-VSA** designs to scale efficiently to large symbol spaces without worst-case memory provisioning.

Additionally, we implemented the retraining part of the baseline training system for comparison. Unlike this approach, the proposed system incorporates the **ID-VSA** framework with a multiscale training strategy operating across three scales. Table VII summarizes the hardware cost differences between the retraining-based baseline and our proposed **ID-VSA** ❷ (`Gaussian Polygon`) design. For the MNIST dataset with D = 1024, the proposed framework maintains competitive classification accuracy while delivering substantial improvements in hardware efficiency. In HV generation (Table V), the proposed method (**ID-VSA** ❸) achieves a 109× reduction in area, 370× reduction in power consumption, and 98× reduction in area-delay product. For the training stage (Table VII), the proposed design further achieves 39× savings in area, 73× in power, and 33× in area-delay product. These results demonstrate that the proposed online learning method can effectively replace conventional η-based iterative retraining, offering a lightweight and hardware-efficient alternative for edge AI. In addition, the **ID-VSA** framework supports parallelism within the `Gaussian Polygon` structure: multiple `MUX` units can be used in parallel to accelerate performance. While this introduces some additional hardware overhead, the resulting gains in throughput make the tradeoff attractive for resource-constrained but high-performance applications.

V. DISCUSSION

A. Scaling Regime: Remote Sensing and 3-D Image Datasets

Motivated by the scaling trends highlighted in Fig. 1 and the requirements of emerging HDC applications, we extend our evaluation beyond small, canonical datasets to more demanding remote sensing and 3-D image workloads. These regimes pose challenges not only in terms of classification accuracy, but also in symbol scalability, as higher spatial resolution and additional dimensions substantially increase the number of position-dependent symbols that must be represented. In particular, 3-D MedMNIST variants introduce an additional spatial axis, effectively multiplying the symbol space and exacerbating the memory and encoding limitations of dictionary-based HDC architectures. To assess performance under these conditions, we evaluate the proposed designs on the 2-D EuroSAT [51] and 3-D FractureMNIST3D [52] (a MedMNIST variant) datasets, with results summarized in Table VIII for D = 1024. As expected, retraining improves baseline performance on both datasets: for FractureMNIST3D, accuracy increases from 42%–50% without retraining to about 59% after retraining, while EuroSAT baselines reach around 80% accuracy. In contrast, the proposed `Gaussian Polygon`-based designs consistently outperform the baselines without retraining, achieving up to 84.69% accuracy and 0.85 $F1$ on EuroSAT, while providing more modest but consistent improvements on FractureMNIST3D. Overall, the results demonstrate that spatial upscaling via `Gaussian Polygon`

TABLE VIII

COMPARATIVE ANALYSIS OF ENCODING STRATEGIES ACROSS 2-D RGB REMOTE SENSING AND 3-D MEDICAL IMAGING TASKS

Encoding Approach ($D = 1024$)	EuroSAT Acc.(%)	EuroSAT F1	FractureMNIST3D Acc.(%)	FractureMNIST3D F1
Baseline-P w/o retrain [35]	77.37	0.78	41.75	0.42
Baseline-P w/ retrain [36]	79.79	0.80	52.50	0.53
Baseline-S w/o retrain [37]	79.81	0.80	49.82	0.50
Baseline-S w/ retrain [32]	80.36	0.80	59.17	0.59
Baseline-H w/o retrain [38]	77.74	0.78	42.32	0.42
Baseline-H w/ retrain [38]	78.88	0.79	58.57	0.59
Baseline-V w/o retrain [39]	77.41	0.78	42.32	0.42
Baseline-V w/ retrain [39]	79.41	0.80	58.75	0.59
Baseline-T w/o retrain-sliding window,✣	74.40	0.74	47.50	0.48
Baseline-T w/ retrain-sliding window,✣	75.87	0.76	54.78	0.55
Baseline-T w/o retrain-sliding window,★	73.60	0.74	47.48	0.47
Baseline-T w/ retrain-sliding window,★	74.73	0.75	54.36	0.54
ID-VSA ❷ (1×)	82.15	0.82	58.83	0.59
ID-VSA ❷ (2×)	83.48	0.83	59.17	0.59
ID-VSA ❷ (4×)	83.48	0.83	59.27	0.59
ID-VSA ❹ (1×)	82.26	0.82	58.88	0.59
ID-VSA ❹ (2×)	83.48	0.83	59.67	0.60
ID-VSA ❹ (4×)	83.48	0.83	59.88	0.60
ID-VSA ❻✣ (1×)	82.15	0.82	57.08	0.57
ID-VSA ❻✣ (2×)	83.36	0.83	59.71	0.60
ID-VSA ❻✣ (4×)	83.36	0.83	59.71	0.60
ID-VSA ❻★ (1×)	84.57	0.84	59.50	0.60
ID-VSA ❻★ (2×)	84.57	0.84	59.60	0.60
ID-VSA ❻★ (4×)	84.69	0.85	59.60	0.60
ID-VSA ❻-sliding window,✣ (1×)	82.27	0.82	57.50	0.58
ID-VSA ❻-sliding window,✣ (2×)	82.27	0.82	59.50	0.60
ID-VSA ❻-sliding window,✣ (4×)	83.36	0.83	59.50	0.60
ID-VSA ❻-sliding window,★ (1×)	83.24	0.83	57.50	0.58
ID-VSA ❻-sliding window,★ (2×)	84.42	0.84	59.50	0.60
ID-VSA ❻-sliding window,★ (4×)	84.42	0.84	59.50	0.60

✣→TRNG1 [33], ★→TRNG2 [34]. EuroSAT uses $(64, 128, 256)$ for $1\times$–$4\times$ upscaling, while FractureMNIST3D uses $(28, 56, 112)$.

yields more consistent accuracy improvement than baseline encodings for these future large-scale HDC deployments.

B. Microbenchmark: Gaussian Polygon Versus Regular HDC

To complement the accuracy comparisons in Tables III and VIII for the runtime and storage analysis, we introduce a lightweight *microbenchmark* that isolates the dominant cost drivers of: 1) symbol construction; 2) inference; and 3) training-time updates when comparing `Gaussian Polygon` with regular HDC pipelines. Here, regular HDC refers to baseline encodings operated either: 1) *without* retraining (single-pass inference) or 2) *with* retraining (iterative prototype refinement). The no-retraining baseline is used as the runtime reference, and all runtime factors are reported as normalized "×" overhead relative to a single-pass native-resolution baseline; dataset I/O is excluded. The microbenchmark is implemented with packed bit-operations (e.g., `XOR`/binding and popcount-based similarity) and decomposes the pipeline into four stages: 1) symbol dictionary generation or access (position and level symbols); 2) encoding; 3) similarity search (Hamming distance to class prototypes); and 4) update rules (prototype accumulation for no retraining, and misprediction-driven update for retraining). To enable a fair comparison, we define an *ISO* target as the accuracy achieved by `Gaussian Polygon` at a given upscale factor, and measure the baseline retraining overhead as the training-time cost required to reach this ISO target under edge constraints [32]. Accordingly, retraining runtime scales with the number of epochs and learning rate tuning iterations needed to achieve the ISO accuracy.

Table IX reports the maximum additional memory demanded by the training pipeline. For a retraining-based baseline, this cost is dominated by the need to store pregenerated position and level symbols throughout training, which scales with the number of pixels and quantization levels. In contrast, `Gaussian Polygon` operates with a tile-streaming working set (corner symbols and a small set of temporary registers used in the `MUX`-based construction), enabling on-the-fly operation without storing the full symbol dictionary. For EuroSAT, the tile-streaming working set scales linearly with the number of channels (RGB), yielding the reported memory of 6.12 KB.

TABLE IX

GAUSSIAN POLYGON VERSUS REGULAR HDC RETRAINING: MEMORY AND TRAINING-RUNTIME SCALING

Dataset	Approach	Acc. (%)	Mem. (KB)	Runtime ×
MNIST	Retrain (Baseline-V)	89.39	$133.12^{\dagger}$	$\sim 49.5\times$
	`Gaussian Polygon` (ISO**)	90.23	$2.04^{\ddagger}$	$\sim 20.4\times$
EuroSAT	Retrain (Baseline-S)	80.36	$557.06^{\dagger}$	$\sim 286.8\times$
	`Gaussian Polygon` (ISO**)	83.48	$6.12^{\ddagger}$	$\sim 85.9\times$

†Baseline retraining requires pre-recorded symbols. ‡`Gaussian Polygon` uses *tile-streaming working memory* as it dynamically operates HVs. ** *ISO* [32] is the accuracy achieved by `Gaussian Polygon` and used as the target for the retraining baseline. Retraining runtime depends on epochs and the learning-rate sweep needed to reach ISO under the edge configuration. All runtimes are normalized to single-pass baseline encoding (w/o retrain) at native resolution; dataset I/O is excluded.

As summarized in Table IX, `Gaussian Polygon` achieves the ISO accuracy targets (90.23% on MNIST, Table III; and 83.48% on EuroSAT, Table VIII) with substantially lower end-to-end pipeline memory and reduced training-time scaling compared to retraining-based baselines. Specifically, the required pipeline memory drops from 133 KB to 2.04 KB on MNIST ($\sim 65\times$) and from 557 KB to 6.12 KB on EuroSAT ($\sim 91\times$). Likewise, training-time scaling reduces from $\sim 49.5\times$ to $\sim 20.4\times$ on MNIST and from $\sim 286.8\times$ to $\sim 85.9\times$ on EuroSAT (both relative to the single-pass w/o-retrain baseline).

VI. CONCLUSION

This work revisited the foundations of hyperdimensional learning by introducing the Independent and Dynamic Vector Symbolic Architecture (**ID-VSA**). Central to our approach is the insight that SC and HDC share a common basis in vector-stream generation, enabling a natural and powerful integration of the two paradigms. Building on this, we introduced lightweight single-source vector generators, quasi-normal distribution-based extensions, and true-random mechanisms that ensure orthogonality, adaptability, and hardware efficiency. We further advanced HDC learning through the `Gaussian Polygon`, a novel multiscale interpolation technique that enriches feature representations directly in the HV domain. We emphasize that `Gaussian Polygon` is not intended to universally replace retraining; rather, it offers a complementary, hardware-aware alternative. Experimental results demonstrate that **ID-VSA** consistently matches or surpasses existing HDC approaches, achieving substantial gains in learning performance and hardware efficiency for edge AI.

REFERENCES

[1] M. Moghadam, A. K. M. Masum, S. Aygun, and M. H. Najafi, "ID-VSA: Independent and dynamic vector symbolic architecture for energy-efficient edge AI," in *Proc. IEEE/ACM Int. Symp. Low Power Electron. Design (ISLPED)*, Aug. 2025, pp. 1–7.

[2] A. Rahimi, P. Kanerva, and J. M. Rabaey, "A robust and energy-efficient classifier using brain-inspired hyperdimensional computing," in *Proc. Int. Symp. Low Power Electron. Design*, Aug. 2016, pp. 64–69.

[3] B. Khaleghi, S. Salamat, A. Thomas, F. Asgarinejad, Y. Kim, and T. Rosing, "SHEARer: Highly-efficient hyperdimensional computing by software-hardware enabled multifold approximation," in *Proc. ACM/IEEE Int. Symp. Low Power Electron. Design*, Aug. 2020, pp. 241–246.

[4] A. Kazemi, M. M. Sharifi, Z. Zou, M. Niemier, X. S. Hu, and M. Imani, "MIMHD: Accurate and efficient hyperdimensional inference using multi-bit in-memory computing," in *Proc. IEEE/ACM Int. Symp. Low Power Electron. Design (ISLPED)*, Jul. 2021, pp. 1–6.

[5] J. Morris, M. Imani, S. Bosch, A. Thomas, H. Shu, and T. Rosing, "CompHD: Efficient hyperdimensional computing using model compression," in *Proc. IEEE/ACM Int. Symp. Low Power Electron. Design (ISLPED)*, Jul. 2019, pp. 1–6.

[6] S. Gupta, M. Imani, B. Khaleghi, V. Kumar, and T. Rosing, "RAPID: A ReRAM processing in-memory architecture for DNA sequence alignment," in *Proc. IEEE/ACM Int. Symp. Low Power Electron. Design (ISLPED)*, Jul. 2019, pp. 1–6.

[7] J. Morris, K. Ergun, B. Khaleghi, M. Imani, B. Aksanli, and T. Simunic, "Hydrea: Utilizing hyperdimensional computing for a more robust and efficient machine learning system," *ACM Trans. Embedded Comput. Syst.*, vol. 21, no. 6, pp. 1–22, Oct. 2022.

[8] D. Kleyko et al., "Vector symbolic architectures as a computing framework for emerging hardware," *Proc. IEEE*, vol. 110, no. 10, pp. 1538–1571, Oct. 2022.

[9] S. Duan, X. Xu, and S. Ren, "A brain-inspired low-dimensional computing classifier for inference on tiny devices," in *Proc. TinyML Res. Symp.*, 2022, pp. 1–8.

[10] A. A. Khan, S. Ollivier, S. Longofono, G. Hempel, J. Castrillon, and A. K. Jones, "Brain-inspired cognition in next-generation racetrack memories," *ACM Trans. Embedded Comput. Syst.*, vol. 21, no. 6, pp. 1–28, Dec. 2022.

[11] A. Alaghi, W. Qian, and J. P. Hayes, "The promise and challenge of stochastic computing," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 37, no. 8, pp. 1515–1531, Aug. 2018.

[12] E. H. Adelson, P. J. Burt, C. H. Anderson, J. M. Ogden, and J. R. Bergen, "Pyramid methods in image processing," *RCA Eng.*, vol. 29, pp. 33–41, Nov. 1984.

[13] S. Aygun, M. H. Najafi, M. Imani, and E. O. Gunes, "Agile simulation of stochastic computing image processing with contingency tables," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 42, no. 10, pp. 3474–3478, Oct. 2023.

[14] J. Liu et al., "A hardware pseudo-random number generator using stochastic computing and logistic map," *Micromachines*, vol. 12, no. 1, p. 31, Dec. 2020.

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

[16] T. J. Baker and J. P. Hayes, "CeMux: Maximizing the accuracy of stochastic mux adders and an application to filter design," *ACM Trans. Design Autom. Electron. Syst.*, vol. 27, no. 3, pp. 1–26, Jan. 2022.

[17] S. Aygun, M. Shoushtari Moghadam, M. Hassan Najafi, and M. Imani, "Learning from hypervectors: A survey on hypervector encoding," 2023, *arXiv:2308.00685*.

[18] D. Kleyko, D. A. Rachkovskij, E. Osipov, and A. Rahimi, "A survey on hyperdimensional computing aka vector symbolic architectures, part I: Models and data transformations," *ACM Comput. Surveys*, vol. 55, no. 6, pp. 1–40, Dec. 2022.

[19] S. Zhang, K. Juretus, and X. Jiao, "Exploring hyperdimensional computing robustness against hardware errors," *IEEE Trans. Comput.*, vol. 74, no. 6, pp. 1963–1977, Jun. 2025.

[20] G. Karunaratne, M. Le Gallo, G. Cherubini, L. Benini, A. Rahimi, and A. Sebastian, "In-memory hyperdimensional computing," *Nature Electron.*, vol. 3, no. 6, pp. 327–337, Jun. 2020.

[21] M. Shoushtari Moghadam, S. Aygun, F. S. Banitaba, and M. H. Najafi, "All you need is unary: End-to-end unary bit-stream processing in hyperdimensional computing," in *Proc. 29th ACM/IEEE Int. Symp. Low Power Electron. Design*, New York, NY, USA, Aug. 2024, pp. 1–6.

[22] S. Duan and X. Xu, "HDCOG: A lightweight hyperdimensional computing framework with feature extraction," in *Proc. IEEE/ACM Int. Symp. Nanosc. Architectures (NANOARCH)*, Nov. 2021, pp. 1–6.

[23] M. Imani et al., "Neural computation for robust and holographic face detection," in *Proc. 59th ACM/IEEE Design Autom. Conf.*, Jul. 2022, pp. 31–36.

[24] L. Spiegelhofer, "Discrepancy results for the van der corput sequence," *Uniform distribution theory*, vol. 13, no. 2, pp. 57–69, Dec. 2018.

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

[26] V. Todorov, T. Ostromsky, I. Dimov, and S. Fidanova, "Optimized quasi-Monte Carlo method based on low discrepancy sequences for sensitivity analysis in air pollution modelling," in *Proc. Commun. Papers Federated Conf. Comput. Sci. Inf. Syst.*, Sep. 2020, pp. 25–28.

[27] K.-H. Kim, P.-S. Shim, and S. Shin, "An alternative bilinear interpolation method between spherical grids," *Atmosphere*, vol. 10, no. 3, p. 123, Mar. 2019.

[28] C. Cao et al., "An improved faster R-CNN for small object detection," *IEEE Access*, vol. 7, pp. 106838–106846, 2019.

[29] K. T. Gribbon and D. G. Bailey, "A novel approach to real-time bilinear interpolation," in *Proc. 2nd IEEE Int. Workshop Electron. Design, Test Appl.*, Perth, WA, Australia, Jan. 2004, pp. 126–131.

[30] Y. Nam et al., "Rhychee-FL: Robust and efficient hyperdimensional federated learning with homomorphic encryption," in *Proc. Design, Autom. Test Eur. Conf. (DATE)*, Mar. 2025, pp. 1–7.

[31] T. Thadewald and H. Büning, "Jarque–bera test and its competitors for testing normality—A power comparison," *J. Appl. Statist.*, vol. 34, no. 1, pp. 87–105, Jan. 2007.

[32] S. Aygun and M. Hassan Najafi, "Sobol sequence optimization for hardware-efficient vector symbolic architectures," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 44, no. 3, pp. 937–950, Mar. 2025.

[33] M. Haahr. (2025). *Pregenerated File Archive-Text Files*. Accessed: Aug. 13, 2025. [Online]. Available: https://archive.random.org/text

[34] J. Kelsey, L. T. A. N. Brandao, R. Peralta, and H. Booth, A Reference for Randomness Beacons: Format and Protocol Version 2, Standard NIST.IR.8213, 2019.

[35] E. Hassan, Y. Halawani, B. Mohammad, and H. Saleh, "Hyper-dimensional computing challenges and opportunities for AI applications," *IEEE Access*, vol. 10, pp. 97651–97664, 2022.

[36] C.-Y. Hsieh, Y.-C. Chuang, and A.-Y.-A. Wu, "FL-HDC: Hyperdimensional computing design for the application of federated learning," in *Proc. IEEE 3rd Int. Conf. Artif. Intell. Circuits Syst. (AICAS)*, Jun. 2021, pp. 1–5.

[37] S. Aygun, M. H. Najafi, and M. Imani, "A linear-time, optimization-free, and edge device-compatible hypervector encoding," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Apr. 2023, pp. 1–2.

[38] M. M. Alam, S. Biderman, J. Holt, T. Oates, A. Oberle, and E. Raff, "A Walsh Hadamard derived linear vector symbolic architecture," in *Proc. Adv. Neural Inf. Process. Syst.*, 2024, pp. 2711–2733.

[39] M. S. Moghadam, S. Aygun, and M. H. Najafi, "No-multiplication deterministic hyperdimensional encoding for resource-constrained devices," *IEEE Embedded Syst. Lett.*, vol. 15, no. 4, pp. 210–213, Dec. 2023.

[40] U. Quasthoff, M. Richter, and C. Biemann, "Corpus portal for search in monolingual corpora," in *Proc. LREC*, 2006, pp. 1799–1802.

[41] Y. Lecun. (2024). *Mnist*. [Online]. Available: https://api.openml.org/d/554

[42] D. Kermany et al., "Identifying medical diagnoses and treatable diseases by image-based deep learning," *Cell*, vol. 172, no. 5, pp. 1122–1131, 2018.

[43] W. Al-Dhabyani, M. Gomaa, H. Khaled, and A. Fahmy, "Dataset of breast ultrasound images," *Data Brief*, vol. 28, Feb. 2020, Art. no. 104863.

[44] D. Anguita, A. Ghio, L. Oneto, X. Parra, and J. L. Reyes-Ortiz, "A public domain dataset for human activity recognition using smartphones," *Esann*, pp. 437–442, 2013.

[45] R. Cole and M. Fanty, "Spoken letter recognition," in *Proc. Workshop Speech Natural Lang.-HLT*, 1990, pp. 385–390.

[46] A. Hernández-Cano, N. Matsumoto, E. Ping, and M. Imani, "OnlineHD: Robust, efficient, and single-pass online learning using hyperdimensional system," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Feb. 2021, pp. 56–61.

[47] İ. Emir Yüksel et al., "TuRaN: True random number generation using supply voltage underscaling in SRAMs," 2022, *arXiv:2211.10894*.

[48] P. Hsiung Huang, R. Yang Lu, Y. Hsien Lin, H. Siang Su, and E. Ray Hsieh, "Filamentary random telegraph noise-based multiple-resistive-state true random number generator for probabilistic hot/cold bits," *IEEE Trans. Electron Devices*, vol. 72, no. 7, pp. 3573–3581, Jul. 2025.

[49] A. Bahador, M. H. Moaiyeri, and R. Ghaderi, "Algorithmically enhanced design of spintronic-based tunable true random number generator for dependable stochastic computing," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 44, no. 3, pp. 961–974, Mar. 2025.

[50] D. Vodenicarevic et al., "Low-energy truly random number generation with superparamagnetic tunnel junctions for unconventional computing," *Phys. Rev. Appl.*, vol. 8, no. 5, Nov. 2017, Art. no. 054045.

[51] P. Helber, B. Bischke, A. Dengel, and D. Borth, "EuroSAT: A novel dataset and deep learning benchmark for land use and land cover classification," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 12, no. 7, pp. 2217–2226, Jul. 2019.

[52] L. Jin et al., "Deep-learning-assisted detection and segmentation of rib fractures from CT scans: Development and validation of FracNet," *eBioMedicine*, vol. 62, Dec. 2020, Art. no. 103106.