IEEE Open Journal of
Circuits and Systems

Received 1 November 2023; revised 14 January 2024 and 10 April 2024; accepted 11 May 2024.
Date of publication 14 May 2024; date of current version 31 May 2024.

Digital Object Identifier 10.1109/OJCAS.2024.3401028

StrideHD: A Binary Hyperdimensional Computing System Utilizing Window Striding for Image Classification

DEHUA LIANG[1], JUN SHIOMI[1] (Senior Member, IEEE), NORIYUKI MIURA[1] (Member, IEEE), AND HIROMITSU AWANO[2] (Member, IEEE)

[1]Graduate School of Information Science and Technology, Osaka University, Suita 565-0871, Osaka, Japan

[2]Graduate School of Informatics, Kyoto University, Kyoto 606-8501, Japan

This article was recommended by Associate Editor M. Rastogi

CORRESPONDING AUTHOR: D. LIANG (e-mail: d-linag@ist.osaka-u.ac.jp)

This work was supported by the Japan Science and Technology Agency (JST), PRESTO, Japan, under Grant JPMJPR22B1.

ABSTRACT Hyper-Dimensional (HD) computing is a brain-inspired learning approach for efficient and fast learning on today's embedded devices. HDC first encodes all data points to high-dimensional vectors called hypervectors and then efficiently performs the classification task using a well-defined set of operations. Although HDC achieved reasonable performances in several practical tasks, it comes with huge memory requirements since the data point should be stored in a very long vector having thousands of bits. To alleviate this problem, we propose a novel HDC architecture, called *StrideHD*. By utilizing the window striding in image classification, *StrideHD* enables HDC system to be trained and tested using binary hypervectors and achieves high accuracy with fast training speed and significantly low hardware resources. *StrideHD* encodes data points to distributed binary hypervectors and eliminates the expensive Channel item Memory (CiM) and item Memory (iM) in the encoder, which significantly reduces the required hardware cost for inference. Our evaluation also shows that compared with two popular HD algorithms, the single-pass *StrideHD* model achieves a 27.6× and 8.2× reduction in inference memory cost without hurting the classification accuracy, while the iterative mode further provides 8.7× memory efficiency. Under the same inference memory cost, our single-pass mode *StrideHD* averagely achieves 13.56% accuracy improvement in comparison with the single-pass baseline HD, which is a similar performance even in comparison with the costly iterative baseline HD models. As an extension, the iterative retraining mode of *StrideHD* averagely provides 11.33% accuracy improvement to its single-pass mode, which can be accomplished in fewer iterations in comparison with the baseline HD algorithms. The hardware implementation also demonstrates that *StrideHD* achieves over 9.9× and 28.8× reduction compared with baseline in area and power, respectively.

INDEX TERMS Hyperdimensional computing, distributed system, memory requirement.

I. INTRODUCTION

THE EMERGENCE of the Internet of Things (IoT) has led to a copious amount of small connected edge devices and systems [1]. Many of these devices need to perform classification tasks such as speech recognition [2], activity recognition [3], and image classification [4]. Though Deep neural networks (DNNs) have provided high accuracy for complex classification tasks, with the scale of DNNs increasing, the high computational complexity and memory requirement of DNNs hinder usability to a broad variety of embedded applications. For example, AlexNet [5] requires 249MB of inference memory and performs 1.5B high precision operations to classify one image. Even applying the hardware-friendly techniques to get the Binarized Neural Networks

(BNNs) [6] or XNOR-Networks [7], still requires expensive computation costs due to the floating-point calculation and backpropagation algorithm during the training. The energy constraint of edge devices hinders them from the real-time training of NN models [8].

Although sending the data to a powerful cloud platform to perform tasks is one of the options, there are still transmission delays and privacy security issues. For example, in health care monitoring, we often require learning algorithms to have real-time control of the patient's daily behavior, speech, and bio-medical sensors. Sending all data points to the cloud, cannot guarantee scalability and real-time response, which is often undesirable due to privacy and security concerns [9]. Hence, for edge devices with limited hardware resources, the demand for a more processing-efficient model is rising.

Brain-inspired hyperdimensional computing (HDC) has been proposed as a computing method that processes cognitive tasks in a more lightweight way [10]. HDC aims at realizing real-time performance and robustness through using strategies that more closely model the human brain [11]. HDC relies on mathematical properties of high-dimensional vector spaces and use high-dimensional distributed representations called hypervectors [12]. HDC works based on the existence of orthogonal hypervectors which can be combined using well-defined vector space operations. The mathematics governing the high dimensional space enables HDC to be easily applied to different learning problems. The first step in HDC is to encode/map data points from the original domain to the high-dimensional space with bit-wise operations. During the training, HDC combines the encoded hypervectors to generate a hypervector representing each class. The classification task at inference performs by searching the similarity of an encoded test hypervector with all trained classes.

However, there are still several remaining challenges in the application of HDC in edge devices: (i) Huge memory cost for the encoding procedure. In most HDC models, they need to represent both the index and value of the input features via hypervectors, which requires a large size of memory blocks. Such a process takes huge occupation of the total inference memory cost, e.g., 96.15% for letters recognition task. (ii) Many HDC algorithms need to be trained on expensive floating-point (FP) hypervectors and perform the inference with costly cosine similarity measurement, which leads to the increase of hardware resource requirement in edge devices. (iii) The hypervector of most HDC is generated holistically. For the hypervector with thousands of dimensions, optimization based on dimension-wise sparsity has been proposed in some prior works. However, feature-wise sparsity should also be under consideration when it comes to image processing tasks. (iv) The widely used retraining procedure in current HDC algorithms requires tens of iterations to get saturated, which leads to long training time.

In this work, we propose a novel HDC system: *StrideHD*. The uniqueness of *StrideHD* is to capture the critical features utilizing window striding method and organize the HDC architecture in a distributed way. After utilizing the window striding to chop the input images, thermometer binarization and max-pooling are applied. The extracted binary features are further encoded to hypervectors with high orthogonality, which enables efficient training/testing in our HDC model. Meanwhile, the hypervectors can be generated without the expensive item memory requirement. The main contributions of our *StrideHD* model are as follows:

1) Successfully eliminated the costly Channel item Memory (CiM) and item Memory (iM) by exploiting a pseudo-random hypervector generation mechanism in the encoder. Besides the traditional dimension-wise sparsification, a feature-wise model sparsification is further proposed for image processing tasks.
2) Compared to two HDC baselines, our single-pass training achieved a 27.6× and 8.2× reduction in memory cost without hurting the accuracy, while the iterative training can further improve 8.7× memory efficiency.
3) Under the same inference memory cost, the classification accuracy of single-pass mode *StrideHD* is averagely 13.56% higher than the baseline HDC models.
4) As an extension, we propose an iterative retraining mode in our *StrideHD*, which averagely provides 11.33% accuracy improvement to its single-pass mode in *DistriHD* [13], which can be accomplished in fewer iterations compared to the other baseline HD models.
5) For hardware cost, we achieved over 9.9× and 28.8× reduction compared with baseline HD [9] in area and power, respectively.

II. RELATED WORK

Hyperdimensional Computing (HDC) leads to fast learning ability, high energy efficiency and acceptable accuracy in learning and classification tasks [14]. For the further improvement of memory efficiency and classification accuracy, many works have been proposed.

The first general idea is to utilize hypervector with high-precision elements. In [15], [16], [17], [18], [19], [20], authors need to encode data points to hypervectors with non-binary elements, i.e., storing integer or FP value for each element. This leads to the high memory requirement and expensive computational cost. To reduce the inference cost, Rahimi et al. [21] and Imani et al. [9] propose binarizing the elements of class hypervectors after the training. Although these approaches can simplify the inference similarity matrix to Hamming distance and lead to a faster computation speed, it comes with the cost of significantly degraded HD classification accuracy on practical image recognition applications. For instance, Imani et al. [19] proposed a HDC model for face image classification task. If the model is binarized, the accuracy performance sharply decreases to 38.9%, which is far lower than the non-binarized mode.

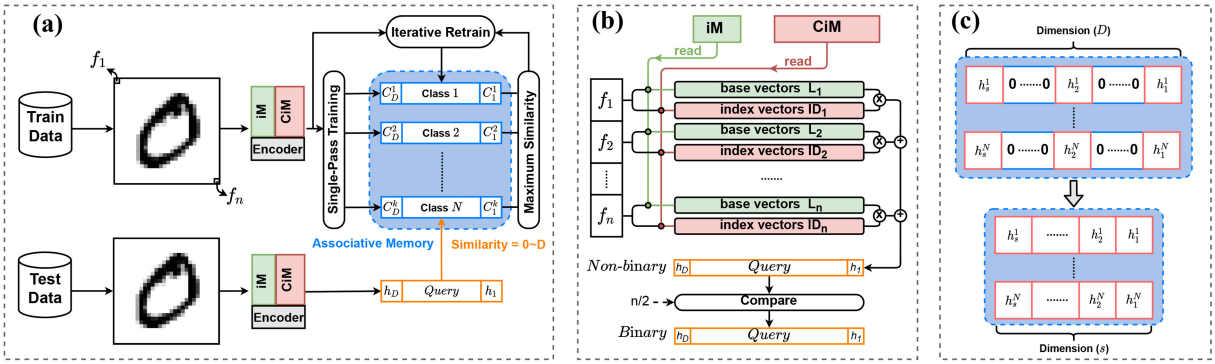

FIGURE 1. (a) Overview of General HDC. (b) Functionality of Record-based Encoder I. (c) Dimension-wise Compression.

To recover the model performance degraded through the quantization step, iterative training algorithms have been proposed [15], [16], [17], [18], [19], [20], [22]. After encoding all the training data point into high-dimensional binary vectors, these hypervectors are stored in the Associative Memory (AM). Utilizing the labeled training data, the similarity between the encoded hypervectors and stored hypervectors of each class can be measured. According to the correctness of prediction, the HD models are adjusted and optimized iteratively. By employing such gradient descent, the error rate of the HD model can be significantly reduced. However, this strategy requires tens of iterations to adjust the model, which leads to a long training time in comparison with the single-pass training.

Since the number of dimensions in the HDC model is strictly related to the performance of classification accuracy, Imani et al. [19] proposed a framework to sparse the HDC model. They explore the prospect of sparsity in hypervectors to improve HDC efficiency without serious loss to accuracy. The research mainly focused on the sparsity of hypervectors class-wise and dimension-wise. Such strategy enables discarding the elements with minimal impact on the results and discarding the inconsequential (non-informative) dimensions shared across all learned hypervectors, which means the number of dimensions in hypervectors can be equally reduced. In this way, the classification accuracy can be improved with the same memory cost.

However, the redundancy of dimensions in hypervectors is not the only limitation. The difference of contributions from each feature should be taken under consideration, especially in the image classification tasks. Most of the current HDC algorithms use a simple encoder, in which all the pixels of images play an equal role in the generation of the non-binary hypervectors. Then it applies the majority voting to get the binary hypervectors as the representation of the input image. In this encoding mechanism, the crucial information from the key patterns could be minified by the noise from the other less important pixels, which may limit the application of HDC in image processing tasks.

III. PRELIMINARY

A. HYPER-DIMENSIONAL COMPUTING

HDC is a computing paradigm involving long vectors with dimensionality in the thousands, which are called hypervectors. In high-dimensional space, there are several nearly orthogonal hypervectors. HD exploits well-defined vector operations to combine these hypervectors, while also preserving most of the information of the hypervectors. Hypervectors are holographic and (pseudo) random with i.i.d. components and full holistic representation, thus no component has more responsibility to store any piece of information than any other hypervector.

Fig. 1(a) shows the overview of the classification in high dimensional space. HD consists of an encoder and an AM. For all sample data within a class, HD maps data to high dimensional vectors, called hypervectors, then combines them together to create a single hypervector modeling each class. Thereafter, the encoded hypervectors belonging to the same prediction class (label) are accumulated to build up the class's hypervector. All trained class hypervectors are stored in the AM. During the inference process, the same encoding scheme maps test input data to high dimensional space. AM looks at the similarity of the generated query hypervector against all stored class hypervectors. The input then gets the label of that class with which it has the highest similarity.

B. GENERAL ENCODING APPROACH

The goal of HDC is to represent the input data in high-dimensional space. In the current existing HDC algorithm, models need to encode the input data to a single hypervector with D dimensions. Aygun et al. [23] summarized many encoding styles for the prior HDC. The difference of encoders epitomizes from two perspectives: (i) The operation in the HD space. E.g., the binding in Record-based encoder, the bundling in Random-projection encoder, and the permutation in N-gram based encoder [21]. (ii) The way to consider the impact of each feature value on the final hypervector. For the reference, we explain the functionality of three popular encoders in detail:

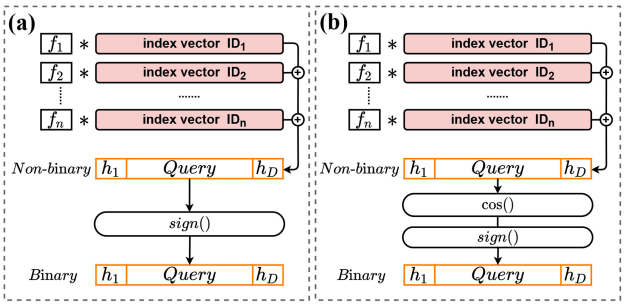

FIGURE 2. (a) Functionality of Encoder II. (b) Functionality of Encoder III.

TABLE 1. Previous HDC encoders comparison.

	Encoder I	Encoder II	Encoder III
CiM	✓	✓	✓
iM	✓	—	—
Arithmetic	XOR int Add sign	— int Add sign	FP Mul FP Add sign & cos

TABLE 2. Inference memory occupations in [9].

	MNIST	ISOLET	UCIHAR	FACE
CiM	94.92%	91.41%	92.73%	94.70%
iM	3.87%	4.74%	5.29%	4.98%
AM	1.21%	3.85%	1.98%	0.31%

Encoder I: Record-based encoder

Fig. 1(b) shows the functionality of this encoding scheme, which was proposed and utilized in [9], [24]. Assume the original data point has n features $\{f_1, \ldots, f_n\}$. The first step is to quantize the range of pixel values into m levels. Then it assigns a random binary hypervector with D dimensions to each quantized level $\{L_1, \ldots, L_m\}$, where L_i is the i^{th} feature values level. The number of dimensions D in the hypervector is large enough compared to the number of features ($D>>n$) in the original data. The level hypervectors are generated such that the neighbor levels have higher similarity, as their absolute values have closer distance. To take the impact of each feature position under consideration, the encoding module assigns a random binary hypervector to each existing feature index $\{ID_1, \ldots, ID_n\}$, where $ID \in \{0, 1\}^D$. These IDs are randomly generated such that all features will have orthogonal IDs. The encoding can happen by linearly combining the feature values over different indices, where a hypervector corresponding to a feature index preserves the position of each feature value in a combined set:

$$H = ID_1 \oplus \overline{L}_1 + ID_2 \oplus \overline{L}_2 + \cdots + ID_n \oplus \overline{L}_n. \quad (1)$$

Here the H is the non-binary encoded hypervector, $\oplus$ denotes the XOR operation, and $\overline{L}_i$ is the binary hypervector corresponding to the i-th feature of vector F. The binarization of the encoded hypervector can happen by comparing each dimension of H with $n/2$ value. All dimensions with a smaller value than $n/2$ are assigned to 0, while other elements are assigned to 1.

Encoder II: Random-projection encoder

Unlike Encoder I, the second encoder differentiates feature positions by multiplying feature values with the corresponding index hypervector, $ID \in \{-1, 1\}^D$ and adding them for all the features. Fig. 2(a) shows the functionality of the second encoding scheme, which is proposed in [16]. For example, where f_i is a feature value, the following equation represents the generation of encoded hypervector H:

$$H = sign(f_1 * ID_1 + f_2 * ID_2 + \cdots + f_n * ID_n) \quad (2)$$

Here the H is the binary encoded hypervector, the $*$ denotes the element-wise multiplication, and $sign$ is a sign function that maps the elements of results to '+1' or '0'.

Since the element in IDs is '-1' or '+1', which simplified the element-wise multiplication of $f_i * ID_i$ to the wire connection in hardware, i.e., using the readout data from the index hypervectors IDs as the sign bit of input features f_i. This encoder also assigns a unique index hypervector ID to each feature position and is stored in the Channel item Memory (CiM). But on the other hand, the item Memory (iM) is eliminated at the cost of broadening the range of summarizing results $\sum_{i=1}^{n} f_i * ID_i$.

Encoder III: Non-linear encoder

Fig. 2(b) shows the functionality of the third encoding scheme, which is proposed in [25]. This method explicitly considers non-linear interactions between input features. Though the data is not linearly separable in original dimensions, it might be linearly separable in higher dimensions. To generate a binary hypervector $H = \{h_1, h_2, \ldots, h_D\}$ with D dimensions from an input feature vector $F = \{f_1, f_2, \ldots, f_n\}$, the first step is to calculate a dot product of the feature vector with a randomly generated vector as $h_i = cos(ID_i \cdot F)$. Here the ID_i represents the index hypervector, which is a randomly generated vector from a Gaussian distribution (mean λ=0 and standard deviation σ=1) with the same dimension as the feature vector. After this, the final encoded hypervector can be obtained by binarization with a sign function.

Note that to ensure the index hypervectors IDs follow the Gaussian distribution and keep the summaries results in a suitable range before applying cosine function, the elements in IDs need to be floating-point data, which seriously increases the memory requirement in the CiM. Meanwhile, in comparison with Encoder II, it requires the more expensive floating-point arithmetic, which limits the computation efficiency during the encoding procedure.

Table 1 shows the comparison between these three popular encoders. The values of level hypervectors Ls and index hypervectors IDs are stored in the item Memory(iM) and the Channel item Memory (CiM) respectively. Take the Encoder I utilized in [9] as an example, for different classification tasks, the occupations of memory cost during the inference are shown in Table 2. We observed that the CiM and iM averagely take more than 93.4% and 4.7% of the inference memory cost, respectively. Meanwhile, the arithmetic difference is seriously affecting the hardware

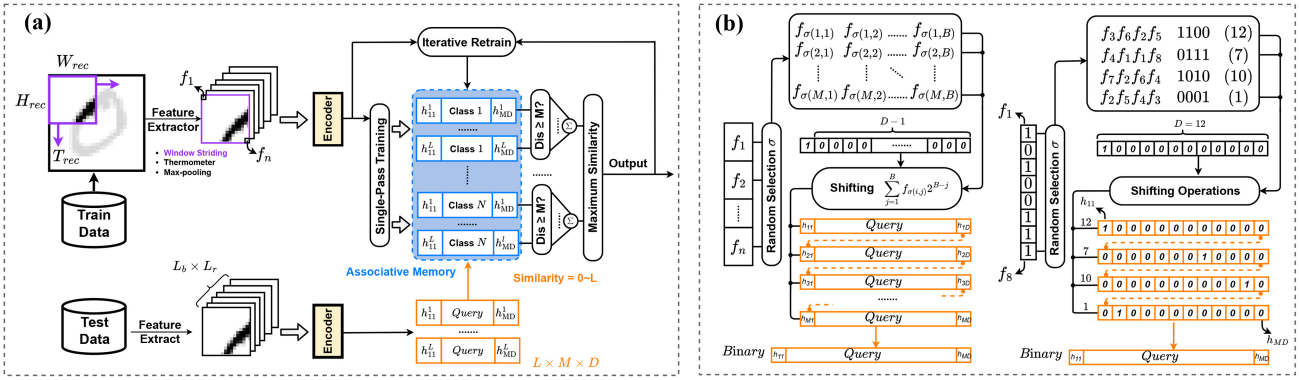

FIGURE 3. (a) Overview of the proposed *StrideHD*. (b) Overview of proposed encoder procedure.

resource requirement, e.g., the FP calculations in Encoder III increase the computational complexity and memory requirement under the same number of dimensions. Hence, we proposed a novel encoding scheme to eliminate the usage of CiM and iM while maintaining the simplicity of arithmetic, which is described in Section IV-E.

C. MODEL SPARSIFICATION

The goal of HDC at inference is to find a class hypervector with the highest similarity to the query hypervector, which is relative to the class hypervectors. However, not all dimensions of the class hypervectors have useful information that can distinguish one class from others. In some of the dimensions, all class hypervectors store common information shared among all classes, which add relatively similar weight to all classes in calculating the Cosine distance or Hamming distance. Reference [19] proposed a framework to explore the sparsity of hypervectors on the class-wise and dimension-wise, which enables discarding the elements with minimal impact on the results and discarding the inconsequential (non-informative) dimensions shared across all learned hypervectors. The number of dimensions in hypervectors can be equally reduced and the classification accuracy can be improved with the same inference memory cost.

For the dimension-wise sparsity in HDC, the changes in the class elements in each dimension should be measured. After obtaining the variation of dimensions, the dimensions with the lowest change are selected to be dropped from the HDC model as they have the least impact on differentiating the classes. Fig. 1(c) shows the overview of sparsification on dimension-wise.

However, the redundancy of dimensions in hypervectors is not the only limitation. The difference of contributions from each feature should be taken under consideration, especially in the image classification tasks. Most of the current HDC algorithms use a simple encoder as we explained in Section III-B, in which all the pixels of images play an equal role in the generation of the non-binary hypervectors. Then it applies the majority voting to get the binary hypervectors as the representation of the input image. In this encoding mechanism, the crucial information from the key patterns could be minified by the noise from the other less important pixels, which may limit the application of HDC in image processing tasks. Hence, we proposed a novel sparsification mechanism to match our *StrideHD* model, which can be applied on the dimension-wise and feature-wise. The details will be described in Section IV-E.

IV. PROPOSED METHOD

As a novel HDC system, *StrideHD* utilizes window striding to capture the critical features in the distributed way, and enables to train/test the model with such distributed binary hypervectors. Fig. 3(a) shows the overview of *StrideHD* for image classification task. In *StrideHD*, the first step is to extract the feature from the original images, in which we apply the window striding technique. Utilizing the receptive window striding, the critical locality patterns can be captured effectively. Then, for features in each receptive window, we apply maxpooling layer and thermometer encoding method to quantize the non-binary values into binary data [26]. After obtaining the binary features for each receptive window, we apply the hypervector encoder to convert features into hypervectors. Note here that, contrary to the conventional HD, a single input image is converted into multiple (i.e., *distributed*) hypervectors. During the initial single-pass training, these distributed hypervectors are combined in a training module in order to create a set of binary hypervector representing each class. The classification is performed by finding the class distributed hypervectors set which has the highest similarity with the test distributed hypervectors set.

Note again that the similarities are respectively computed for each hypervectors. Further, for the model size reduction, we prune part of the class distributed hypervector by using validation dataset whose procedure is detailed in Section IV-E. Since *StrideHD* works with a binary model, the inference can be performed with hardware-friendly Hamming distance as the similarity matrix. As an extension, we also proposed the iterative learning in our framework for performance improvement. The pseudo-code for the *StrideHD* is further shown as the Algorithm 1 and the

Algorithm 1 *StrideHD* Computing Framework

Design Parameters: shape of striding window $W_{rec} \times H_{rec}$, number of training subsets L_e, number of dimension-wise selected distributed hypervectors M_s, number of feature-wise selected distributed hypervectors L_s, thermometer binarized levels L_b, number of dimensions D.
Require: Initialize an integer model $\mathbb{C}$ and a binary model $\mathbb{B}$ in shape of $N \times M \times L \times D$ with '0's, where N is the number of categories, M is the number of dimension-wise distributed hypervectors, L is the number of feature-wise distributed hypervectors.
Ensure: Update the binary model $\mathbb{B}=sgn(\mathbb{C})$
{① Single-pass & Iterative Training}
Split training set to L_e minibatch including samples u^e with $N_{feature}$ non-binary features $u(n)$, which corresponds to the labels $y(u)$.
$u^{(e)}(n) \leftarrow u(n)$, $e=1, 2, ..., L_e$
for $e=1$ to L_e **do**
  for $n=1$ to $N_{feature}$ **do**
    # thermometer binarization
    $u^{(e,b)}(n) \leftarrow u^{(e)}(n)$, $b=1, 2, ..., L_b$ using Eq.(3)
    for $b=1$ to L_b **do**
      # window striding & max-pooling
      $u^{(e,b,l)}(n') \leftarrow u^{(e,b)}(n)$, $l=1, ..., L_b$; $n'=1, ..., W_{rec} \times H_{rec}$
Reshape $u^{(e,b,l)}(n')$ as f_n^l, $l \in [0, L]$, $n \in [1, W_{rec} \times H_{rec}]$
where $L=L_e \times L_b \times L_r$
Generate a random matrix $R_{(i,j)} \in [1, W_{rec} \times H_{rec}]^{M \times B}$
for $i=1$ to M **do**
  $p_i \leftarrow \sum_{j=1}^{B} f_{R_{(i,j)}} 2^{B-j}$ using Eq.(5)
  for $l=1$ to L **do**
    if *single-pass training* **then**
      for $k=1$ to N **do** *# all categories*
        $\mathbb{C}_k^l[p_i] \leftarrow \mathbb{C}_k^l[p_i]+1$
    else if *iterative training* **then**
      if $y_{predict} \neq y$ **then**
        $\mathbb{C}_{correct}^l[p_i] \leftarrow \mathbb{C}_{correct}^l[p_i]+1$ *# correct category*
        $\mathbb{C}_{predict}^l[p_i] \leftarrow \mathbb{C}_{predict}^l[p_i]-1$ *# predict category*

Algorithm 2 Sparsification & Inference

{② Sparsification}
Calculate the accuracy of distributed hypervectors $\mathbb{C}^l[p_i]$ where $i \in [0, M]$, $l \in [0, L]$, discard the $\mathbb{C}^l[p_i]$ with poor Acc.
Model $\mathbb{C}$ and $\mathbb{B}$ are compressed as: $M \rightarrow M_s$, $L \rightarrow L_s$
{③ Inference/Prediction}
for $k=1$ to N **do**
  for $l=1$ to L_s **do** *# before* ②:L; *after* ②:L_s
    $Similarity(k)$ += AND$(\mathbb{C}_k^l[p_i])$, $i=1, 2, ..., M_s$
    # before ②:M; *after* ②:M_s
Output ← $Argmax(Similarity)$

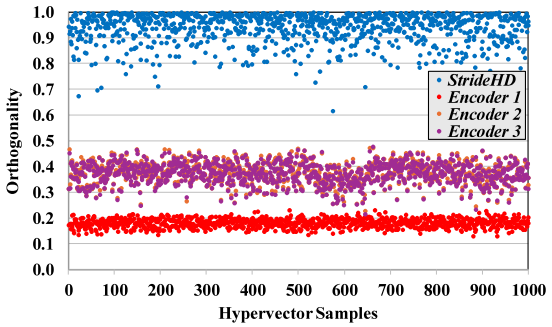

FIGURE 4. Hypervector Orthogonality of Encoders.

Algorithm 2. In the following, we explain the details of the *StrideHD* functionality.

A. FEATURE EXTRACTION BY WINDOW STRIDING

For the input image, we firstly apply a $W_{rec} \times H_{rec}$ size receptive field with a stride of T_{rec}, i.e., chopping the pixels with striding windows. Then we apply a max-pooling operation to improve the generalization of the model. After these two steps, L_r distributed patterns with non-binary elements are generated. Subsequently, we binarize the non-binary element with thermometer way [26]. Assume the range of non-binary element u is $\Delta u = u_{max} - u_{min}$ and quantize it into L_b levels as follow:

$$u^{(b)} = \begin{cases} 1, & \frac{\Delta u \cdot b}{L_b} < u, \\ 0, & \frac{\Delta u \cdot b}{L_b} \geq u. \end{cases} \tag{3}$$

where $b \in [1, L_b]$. In this way, $L_r \times L_b$ distributed binary patterns are extracted from each image.

B. ENCODER

Considering the expensive memory cost of item Memory, we proposed an encoder with a pseudo-random hypervector generation mechanism that can map a pattern to a hypervector only using the shifting operation. Fig. 3(b) shows the overview of the proposed encoder, and Fig. 4 shows the orthogonality comparison between the hypervectors generated by different encoders. Assuming the l-th binary pattern ($l \in [1, L_b \times L_r]$) obtained via feature extraction is represented by vector F^l. Where $F^l = \{f_1, f_1, \ldots, f_n\}$ with n elements ($f_i \in \mathbb{N}$), will be mapped to the distributed hypervector $H^l = \{h_1^l, h_2^l, \ldots, h_D^l\}$ with D dimensions ($h_i \in \{0, 1\}^D$). Firstly, we randomly select the elements from vector F^l and construct a matrix R:

$$R = \begin{pmatrix} f_{\sigma(1,1)} & f_{\sigma(1,2)} & \cdots & f_{\sigma(1,B)} \\ f_{\sigma(2,1)} & f_{\sigma(2,2)} & \cdots & f_{\sigma(2,B)} \\ \vdots & \vdots & \ddots & \vdots \\ f_{\sigma(M,1)} & f_{\sigma(M,2)} & \cdots & f_{\sigma(M,B)} \end{pmatrix} \tag{4}$$

where $f_{\sigma(i,j)} \in \{f_1, f_1, \ldots, f_n\}$, $i \in [1, M]$, $j \in [1, B]$. M and B represent the number and range of the generated binary numbers, respectively. And the function σ represents the random selection. Each row of the matrix R can be considered as a binary number p_i:

$$p_i = \sum_{j=1}^{B} f_{\sigma(i,j)} 2^{B-j} \tag{5}$$

Similar to Bloom filter [26], we can generate M binary vectors in the range of B-bits by flipping the p_i-th bit of the empty vector (all logic "0"), which can be easily accomplished with shifting operation. Finally, the distributed hypervector H^l is the connection of these binary vectors.

Such a mechanism has three main advantages as compared to the other encoding methods in the HDC algorithms [9], [16], [17], [18], [19], [20], [21]. First, unlike existing approaches which need to read the index hypervectors from item Memory, this encoding method can generate hypervectors only using shifting operation.

The expensive memory cost by item Memory (shown in Table 2) can be avoided. Second, this method doesn't need to do accumulation and majority voting for each dimension during encoding, which means it is hardware-friendly and suitable for parallel implementation. Third, the hypervectors generation with high orthogonality. We randomly select 1,000 training images from MNIST dataset and generate the hypervectors with different encoders. As shown in Fig. 4, the orthogonality measure is based on Hamming distance, while our encoder achieved superior performance.

C. SINGLE-PASS TRAINING

Initially the AM is blank, i.e., all the values are set to logic "0". After the feature extraction, $L_r \times L_b$ distributed binary patterns are obtained. By using the encoder we described in Section IV-B, a set of distributed hypervectors H^l are generated ($l \in [1, L_b \times L_r]$). For all input within the same class, the training data will be divided into L_e training subset. For all the data within the same training subset $\{S_1, \ldots, S_K\}$, the distributed hypervectors are added to create the class distributed hypervectors $\mathbb{C} = \{\mathbb{C}^1, \mathbb{C}^2, \ldots, \mathbb{C}^L\}$ with integer addition for each dimension as follow:

$$\mathbb{C}^l = H_1^l + H_2^l + \cdots + H_K^l \tag{6}$$

where $\mathbb{C}_i^l$ represents the l-th distributed hypervector belonging to the i-th class and $L = L_b \times L_r \times L_e$. These class distributed hypervectors will be stored in the AM during the training. Note that in comparison with the existing HD model, the majority voting is simplified to OR operation, which contributes to a fast and ultra-efficient learning process.

D. INFERENCE

After the feature extraction and encoding procedure, L distributed hypervectors H^l are generated to represent one input image for query. Similarly, L distributed class hypervectors $\mathbb{C}^l$ have been trained and stored in the AM part, which contains the universal information of each class in HD space. When it comes to the inference, we measure the similarity of them as follows:

$$Similarity = \sum_{l=1}^{L} sgn\Big[\delta\Big(H^l \& \mathbb{C}^l, \mathbb{C}^l\Big) - M\Big]. \tag{7}$$

Where & represents the bitwise AND operation and the δ is the Hamming distance between two vectors. The *sgn* represents the sign function that extracts the sign of a real number. The class of distributed hypervectors with the highest similarity is chosen as a representative category of the testing image.

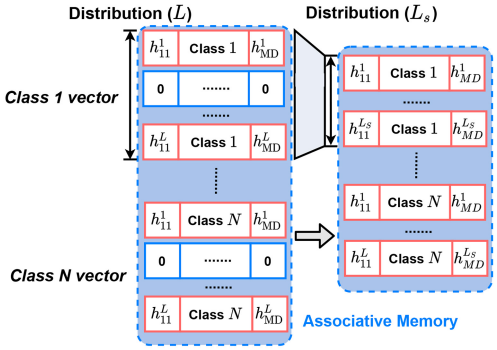

FIGURE 5. Overview of Feature-wise Compression.

E. OPTIMIZED MODEL SPARSIFICATION

To improve memory efficiency, we proposed to sparsify associative memory with the labeled validation data. Compared with the existing mechanism, our model sparsification is changed from unsupervised to supervised mode, which are performed dimension-wise and feature-wise. The details are shown in Fig. 1(c) and Fig. 5.

For dimension-wise sparsity, we use the labeled validation images to calculate the accuracy of each dimension. In the ideal scenario, the value of a dimension in the validation hypervector should be equal to the corresponding dimension in the matching class hypervector, while all other values of this dimension in the mismatching class hypervectors should not be equal to it. After obtaining the accuracy performance of dimensions, the dimensions with the lowest accuracy are selected to be dropped from the HDC model as they have the least/worst impact on differentiating the classes. Note that in our proposed encoding, the distributed hypervector H^l is a connection of the shifted binary vectors, which means the dimensions in these shifted binary vectors can not be dropped separately. Hence, the discard of dimensions only happened for the whole shifted binary vectors, which makes the dimensions of class distributed hypervectors decrease: $\mathbb{C}^l \in \{1, 0\}^{M \cdot D} \rightarrow \mathbb{C}^l \in \{1, 0\}^{M_s \cdot D}$.

Similarly, for the feature-wise sparsity, we can also calculate the accuracy for class distributed hypervector by validation images simultaneously. The $\mathbb{C}^l$ with the lowest accuracy are selected to be dropped as those represented features have the least impact on differentiating the classes. The number of $\mathbb{C}^l$ is decreased: $l \in [0, L] \rightarrow l \in [0, L_s]$.

F. ITERATIVE RETRAINING

As an extension, we also proposed iterative learning in our framework, which aims at reducing the error rate of the initial HD model by employing gradient descent. As shown in Figure. 3(a), the *StrideHD* firstly encodes the training data to query distributed hypervectors, then check its similarity with each pre-stored class distributed hypervector set as mentioned in Section IV-D. If the class distributed hypervector set with the highest similarity matches the correct label, the *StrideHD* ignores updating the model. However, if an encoded training data H incorrectly matches

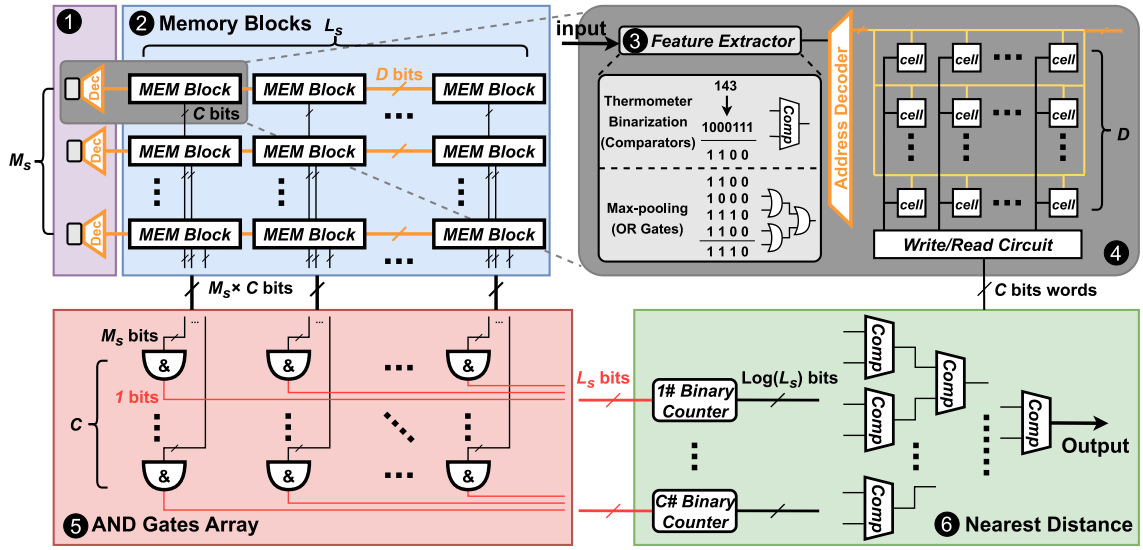

FIGURE 6. The hardware implementation of *StrideHD* during inference includes Encoding, Associative Memory Blocks, AND Gates Array, and Nearest Distance Searching modules.

with the model, we add this query to the correct class $\mathbb{C}_{correct}$ while subtracting it from the predicted incorrect class $\mathbb{C}_{predict}$ as follows:

$$\begin{cases} \mathbb{C}_{correct} = \mathbb{C}_{correct}[+]H \\ \mathbb{C}_{predict} = \mathbb{C}_{predict}[-]H \end{cases} \tag{8}$$

Where $[+]$ and $[-]$ is a binary addition and subtraction for each dimension. Note that such accumulation is performed in class distributed hypervectors with integer elements in a pre-defined range, which is also considered as the counters model [9]. When it comes to the inference, we can use an additional binarized model with the same size of dimensions. All dimensions with a smaller value than 0 are assigned to 0, while other elements are assigned to 1.

Based on the novel encoding procedure described in Section IV-B, the percentage of logic '1' in the encoded hypervector of *StrideHD* is much less than the conventional hypervector. Hence, the update of model mainly focuses on the more significant dimensions, which leads to efficient iterative retraining and fewer iterations.

V. HARDWARE IMPLEMENTATION OF INFERENCE

The hardware implementation of *StrideHD* during inference mainly consist of four different blocks: (❶) Encoding, (❷) Associative Memory Blocks, (❺) AND Gates Array, and (❻) Nearest Distance Searching modules. Figure 6 shows the overview of our hardware architecture.

A. ENCODING BLOCKS

The hardware implementation of each encoding block is shown in Figure 6 (❹). This process performs the (❸) feature extraction and pseudo-random hypervectors generation mechanism with the memory address decoder, which are integrated in the (❶) as hardware implementation of the Encoding Blocks. Mathematically, the computation of feature extraction can be performed by the window striding, thermometer binarization, and max-pooling techniques.

- As for the window striding, it is mainly implemented with the wire connection, which does not require additional transistors in hardware implementation.
- The next step is to convert each original decimal feature to the L_b-bits thermometer binary data. This process can be accomplished with cheap combinational logic circuits. According to Eq.(3), the more continuous '1's at the beginning of the converted data represent the larger value of the original feature.
- Based on the characteristic of thermometer binary data, it is convenient for the implementation of the max-pooling layer, which can be performed by bit-wise OR gates for each bit of the binary data.

Figure 6(❸) shows an example of thermometer binarization and its corresponding max-pooling operation. After this step, each original data point is converted to L_s different M_s-bits binary patterns, which are the input of the address decoders. Based on the decoded addresses, the data from the corresponding memory cells in C different categories are read. Note that the L_s and M_s represents the number of feature-wise and dimension-wise distributed hypervectors after the model sparsification, which has been accomplished by validation data during the training. Such model sparsification process aims at further reducing the memory usage of well-trained model, leading to a light weight requirement for hardware implementation during the inference.

B. ASSOCIATIVE MEMORY (AM) BLOCKS

Figure 6(❷) shows the implementation of the AM Blocks. Unlike the prior HDC algorithms that read the data from all the memory cells, the *StrideHD* only requires reading parts of the memory cells. For each inference operation, $L_s \times M_s \times C$

bits of data are read from the AM Blocks, while the AM Blocks contain D times larger memory in total. Based on the decoded addresses, the data from the corresponding memory cells are read, which leads to M_s different C-bits data as the input of the following AND gates array. Note that the readout data are not the hypervector of each category but represent the similarity measure results.

C. AND GATES ARRAY

Figure 6(❺) shows the implementation of the AND gates array. Compared to the XOR gates array in prior works, even costing the same number of memory cells in AM Blocks, the scale of AND gates array in *StrideHD* is much smaller. On one hand, the scale of readout data from AM is reduced for D times. On the other hand, the *StrideHD* are checking whether the readout dimensions are '+1', while the traditional way (Encoder I, II, and III) requires to check whether the readout dimensions are equal to query hypervector. Hence, readout data from AM blocks are the only input of AND gates array, while the two inputs of the traditional XOR gates array are the readout data and query hypervector, respectively. When M_s-bits of readout data from the same category are '+1', the similarity of the corresponding category is increased by 1, which makes the similarity in the range from 0 to L_s.

D. NEAREST DISTANCE SEARCHING

Figure 6(❻) shows the implementation of the nearest distance searching module. After getting $C \times L_s$ bits data from AND gates array, it requires C different binary counter to calculate the number of '+1' in the L_s-bits data, which is considered the similarity for each category. Finally, utilizing the comparators to get the output. The range of similarity in *StrideHD* is much smaller than the traditional one ($D >> L_s$).

VI. EXPERIMENT

A. EXPERIMENTAL SETUP

We consider the popular HDC algorithm [9] as the baseline, which is similarly utilizing binary hypervectors and eliminating FP calculations. We evaluated *StrideHD* and baseline HD training and inference with three encoders on an Intel Core i7 7600 CPU using an optimized C++ implementation. To verify recognition quality of *StrideHD*, we consider three problems: MNIST [4], Kuzushiji-MNIST [27], Fashion-MNIST [28], and SMILES [29]. For the MNIST-kind datasets, we randomly select 55k images for training and 5k images for validation.

MNIST: is a collection of handwritten digits in grayscale format and is intensively used to compare the performance of many classification models.

Kuzushiji-MNIST: is a dataset that focuses on Kuzushiji (cursive Japanese) [27]. Even though this dataset is created as a drop-in replacement for the MNIST dataset, the characteristics of Kuzushiji and Arabic numbers are completely different, which makes it more challenging than MNIST.

Fashion-MNIST: is a new dataset comprising of fashion products, such as shirts, T-shirts, or coats that look very similar at 28×28 pixel resolution in grayscale, making many samples ambiguous even for humans (Human performance on Fashion-MNIST is only 83.5% [28]).

SMILES: is a face recognition task that aims at classifying the images with or without smiling. There are 13,165 images in the dataset, with each image having a size of 64×64 pixels. Among all the face images, 9475 of these examples are not smiling, while only 3,690 belong to the smiling class. Hence, we randomly select 600 positive images 600 negative images for testing, and 300 images for the validation.

In software, we utilize the MNIST dataset to explore the impact of several design parameters in *StrideHD* single-pass training, and the reduction of the inference memory cost are evaluated in comparison with the baseline HD. As an extension, we make a comparison between our proposed *StrideHD*, and three baseline HD algorithms. The BinHD, SecureHD, and DUAL represent the iterative HDC learning framework in [9] utilizing the Record-based Encoder I from [9], Random-projection Encoder II from [16], and Non-linear Encoder III from [25], respectively.

B. SINGLE-PASS TRAINING PARAMETERS TUNING

In our experiment for MNIST dataset, the max-pooling layer is selected to have a stride of 2, a squared window of size 2, and zero padding. According to the evaluation results shown in Fig. 7(a), we applied a 5×5 receptive window with a stride of 3. Meanwhile, Fig. 7(b-c) illustrates the impact of binary levels L_b and the number of training subsets L_e. With higher L_b and L_e, the classification accuracy is improved with a larger memory cost. Hence, for competitive accuracy and memory efficiency, we choose $L_b = 4$ and $L_e = 2$ as a performance trade-off. Fig. 7(d-f) shows the classification accuracy under different Feature-wise sparsity, Dimension-wise sparsity, and the dimensions D. Similarly, we adopt the Feature-wise and Dimension-wise sparsity as 20% and 95%, respectively. The parameters setting of our *StrideHD* in different tasks are listed in Table 3.

C. MEMORY EFFICIENCY

As we mentioned in Table 2, the iM and CiM are averagely taking huge occupation of memory cost (93.4% and 4.7%) during inference for the BinHD, which is unnecessary in the *StrideHD*. Based on the observation of Table 1 and Table 2, we found that the hardware cost for Encoder III in DUAL is much higher than the Encoder I in BinHD and Encoder II in SecureHD due to the expensive FP calculation and high data precision. Hence, we make a comparison of *StrideHD*, BinHD, and SecureHD in terms of classification accuracy and memory cost, which is shown in Fig. 8. For a comprehensive and fair comparison, we evaluated the performance of the baselines in two different ways.

The first way is to calculate the inference memory cost including the CiM and iM part, which performs very poor accuracy in Fig. 8 (a). To achieve the same level of accuracy (e.g., 94.8%), the iterative BinHD and SecureHD require 27.6× and 8.2× memory cost compared to the single-pass

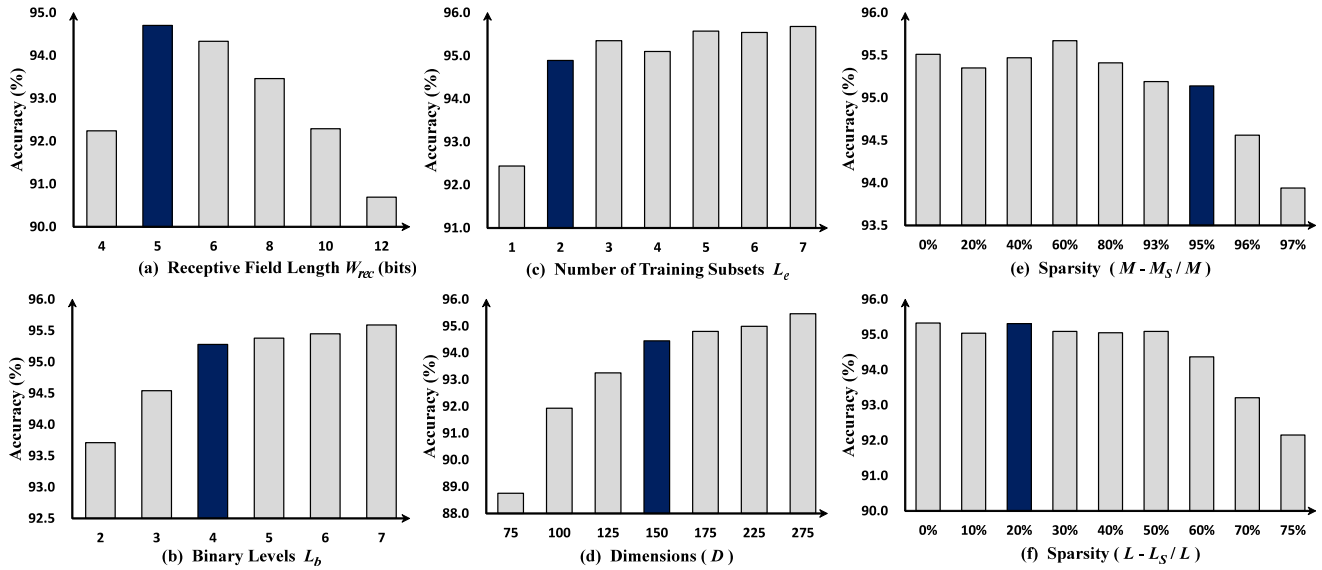

FIGURE 7. Impact of Different Parameters: (a) Length of Receptive Window. (b) Binary Levels L_b. (c) Number of Training Subsets L_e. (d) Dimensions D. (e) Dimension-wise Sparsity. (f) Feature-wise Sparsity.

TABLE 3. Parameters setting.

	W_{rec}	T_{rec}	L_b	L_e	D	S_D	S_F
*mnist	5	3	4	2	150	95%	20%
smiles	7	5	4	2	150	95%	20%

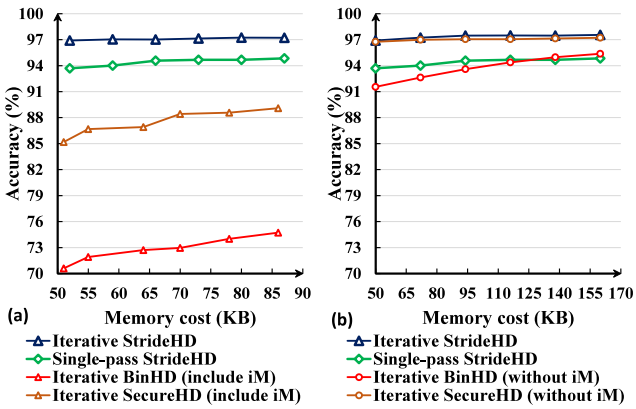

FIGURE 8. The comparison of accuracy and memory cost.

training *StrideHD* model. When applying iterative learning to our proposed model, the accuracy is improved compared to the single-pass training, which results in 8.7× memory efficiency. The significant memory reduction mainly comes from the elimination of the costly CiM and iM.

The second way is to calculate the baseline memory cost without the expensive CiM and iM parts, which can give a fair comparison to the performance of the trained classifiers. Although increasing the number of dimension D in BinHD and SecureHD results in improving the classification accuracy, but also leads to a huge usage of memory during inference. In MNIST dataset, when D is increased to 10K, the accuracy of the iterative baseline HDC models gets saturated at around 96%, which consumes 97.7 KB for the associative memory (AM) part and 7.7 MB for the total memory cost. We found that with the same usage of AM, our single-pass training has the advantage over the iterative BinHD algorithm but is not competitive with the iterative SecureHD. When we apply iterative learning to our proposed model, the accuracy of *StrideHD* is improved to the same level as SecureHD. Hence, such experimental results show that our method successfully eliminated the expensive CiM and iM while maintaining the same accuracy as the classifier.

D. RETRAINING ITERATIONS & CLASSIFICATION ACCURACY

Besides the huge memory efficiency, the fast training process is also considered as the advantage of *StrideHD*. Fig. 9 shows the comparison during the iterative training. The parameters setting of *StrideHD* is shown in Table 3, while the $D = 10k$ for the baseline HDC models. Compared to the baseline HD algorithms, our method requires much fewer iterations to achieve saturated and stable classification accuracy. Since the percentage of logic '1' in the encoded hypervector of *StrideHD* is much less than the conventional hypervecotor in baseline HD algorithms [9], [16], [25]. Therefore, the update of the model can mainly focus on the more significant dimensions, which leads to efficient iterative retraining and fewer required iterations. Meanwhile, most traditional HDC algorithms simply calculating the Hamming distance during the inference query. Although the iterative learning mode of HDC tends to increase the similarity between the class hypervectors and training hypervectors, some of the dimensions still changes back and forth. The noisy information within each training hypervector are accumulated within such dimensions, which might result

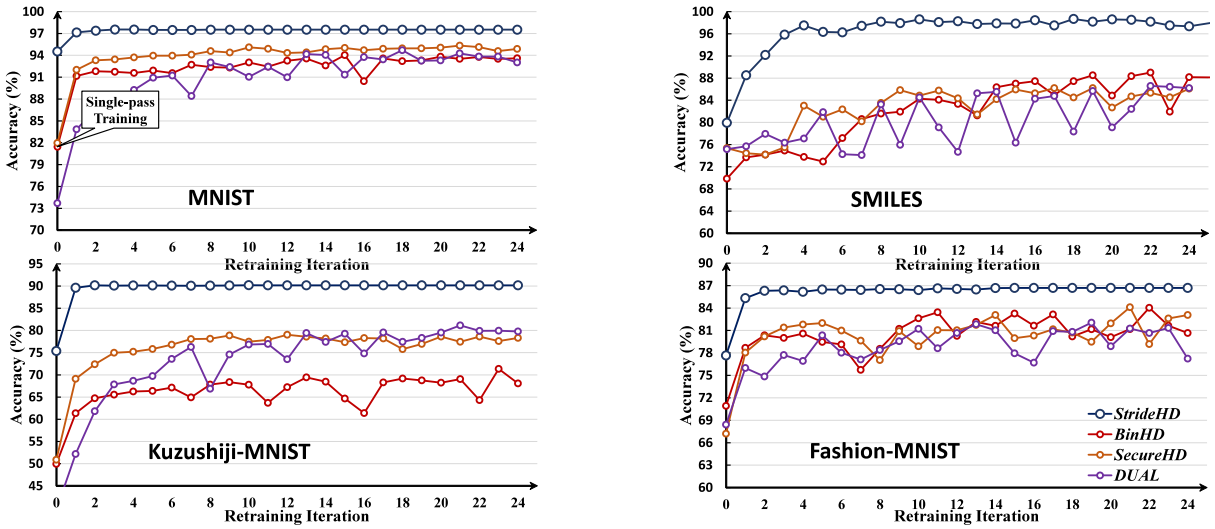

FIGURE 9. Comparison of Iterative Retraining for *StrideHD* and Baseline HD models in different datasets.

TABLE 4. Classification accuracy comparison.

		MNIST	K-MNIST	F-MNIST	SMILES	Average
Single-Pass Mode	***BinHD***	80.48%	49.41%	70.54%	69.83%	67.57%
	SecureHD	81.42%	51.01%	67.06%	75.33%	68.71%
	DUAL	81.25%	51.16%	66.87%	75.17%	68.61%
	StrideHD	**94.52%**	**75.34%**	**77.65%**	**79.92%**	**81.86%**
Iterative Mode	***BinHD***	93.57%	69.16%	82.62%	88.17%	83.38%
	SecureHD	94.87%	78.31%	83.06%	86.09%	85.58%
	DUAL	93.01%	79.76%	81.32%	86.17%	85.07%
	StrideHD	**97.51%**	**90.14%**	**86.67%**	**98.42%**	**93.19%**

in the performance fluctuations of the HDC model. For our *StrideHD*, the bit-wise AND operation and sign function are included during hypervector query, which provides a threshold to reduce the noisy information during the retraining.

For the single-pass training mode, the classification accuracy of *StrideHD* is averagely 13.56% higher than the baseline HDC models. As an extension, the iterative retraining procedure averagely provides an 11.33% accuracy improvement to the single-pass *StrideHD* model. The comparison of classification accuracy is shown in Table 4.

Overall, in comparison with the popular HDC models BinHD, SecureHD, and DUAL, our *StrideHD* significantly reduced the memory cost for the inference by the elimination of costly item Memory. Meanwhile, the fast and simplicity of the training process avoids expensive iterative training while maintaining the same level of classification accuracy.

E. HARDWARE IMPLEMENTATION

The hardware architecture and functionality of *StrideHD* and BinHD are designed via RTL SystemVerilog. Then we use *Synopsys Design Compiler* to synthesize and report the area and power consumption in 65-nm ASIC flow. All the synthesis are based on minimum hardware area cost approach, and the clock period are set as 5 nanoseconds. The memory part can be individually simulated with *CACTI* [30]. The memory type is also chosen to be the main memory in 65-nm ASIC flow, which doesn't contain any tag array and every access will happen at page granularity.

TABLE 5. Hardware performance comparison.

	Baseline	*StrideHD*	Reduction
Area (mm^2)	31.88	3.22	9.9×
Power (mW)	3839	133.2	28.8×

Table 5 shows the comparison of *StrideHD* and the BinHD during inference in terms of ASIC area and power consumption with the same level of classification accuracy (94.8%). The maximum propagation delay of *StideHD* and BinHD is 4.61 and 4.83 nanoseconds, respectively. There are no timing violations. Both *StrideHD* and baseline are using the binary mode, which mostly exploits the hardware-friendly Hamming distance for similarity measurement.

The memory block takes over 96.9% of the area and 99.1% of the power consumption in the *StrideHD* model. The gap in hardware cost between the proposed model and the baseline mainly comes from memory efficiency. The key concept behind *StrideHD* is to eliminate the costly memory blocks in HDC, i.e., the CiM and the iM blocks, which provides significant energy consumption reduction. We aim for this HDC model to bring benefits to customers, irrespective of the type of memory used in hardware implementation. Hence, to exclude the performance gap of the memory, we didn't manually design the memory

part for *StrideHD* and synthesize it alone with the logic parts, but simulate it by the architectural simulation model *CACTI* individually. Both of the baseline HD architecture and our *StrideHD* are compared using the same method to evaluate the power/area performance of the memory block, the accuracy of *CACTI* does not affect the comparison of the hardware cost. Another reason is that the baseline requires the majority voting mechanism to generate the hypervectors while the encoding in *StrideHD* only requires shifting and OR operations, which also contributes to the improvement of hardware efficiency.

VII. CONCLUSION

In this work, we proposed a novel HDC system *StrideHD* that utilizes window striding to capture the locality feature of images. This framework supports using binary hypervectors and achieves high accuracy with fast training speed and significantly low hardware cost. Compared to the baseline BinHD and SecureHD utilizing iterative learning strategy, our framework achieves a 27.6× and 8.2× reduction in memory cost without hurting the accuracy in single-pass mode, while the iterative training can further provide 8.7× memory efficiency. Under the same inference memory cost, the accuracy of single-pass mode *StrideHD* is averagely 13.56% higher than the baseline HDC. As an extension, the iterative retraining mode of *StrideHD* averagely provides 11.33% accuracy improvement to its single-pass mode, which can be accomplished in fewer iterations compared to the baseline HDC. Our hardware evaluation results demonstrate that compared to BinHD, our *StrideHD* achieves over 9.9× and 28.8× reduction in area and power, respectively.

The experiment result illustrates *StrideHD* successfully eliminates the expensive Channel item Memory (CiM) and item Memory (iM) that is frequently used in most HDC algorithms. Compared to the general way, the distribution of hypervectors based on features not only helps the encoder to replace the iM with shifting operation but also constrains the noisy information from the less important pixels or binary channels. Due to the memory efficiency and competitive accuracy, *StrideHD* shows promising potential for image classification tasks aimed at hardware implementation.

REFERENCES

[1] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the Internet of Things," in *Proc. 1st MCC Workshop Mobile Cloud Comput.*, 2012, pp. 13–16.

[2] D. Dua and C. Graff. "UCI machine learning repository." 2017. [Online]. Available: http://archive.ics.uci.edu/ml/datasets/ISOLET

[3] D. Anguita, A. Ghio, L. Oneto, X. Parra, and J. L. Reyes-Ortiz, "Human activity recognition on smartphones using a multiclass hardware-friendly support vector machine," in *Proc. 4th Int. Workshop AAL*, 2012, pp. 216–223.

[4] Y. LeCun, C. Cortes, and C. Burges, "MNIST handwritten digit database: ATT Labs," 2010. [Online]. Available: http://yann.lecun.com/exdb/mnist

[5] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. 25th Int. Conf. Neural Inf. Process. Syst.*, 2012, pp. 1097–1105.

[6] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or −1," 2016, *arXiv:1602.02830*.

[7] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "XNOR-Net: ImageNet classification using binary convolutional neural networks," in *Proc. 14th Eur. Conf. Comput. Vis. (ECCV)*, 2016, pp. 525–542.

[8] C.-Y. Chang, Y.-C. Chuang, C.-T. Huang, and A.-Y. Wu, "Recent progress and development of hyperdimensional computing (HDC) for edge intelligence," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 13, no. 1, pp. 119–136, Mar. 2023.

[9] M. Imani, J. Messerly, F. Wu, W. Pi, and T. Rosing, "A binary learning framework for hyperdimensional computing," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, 2019, pp. 126–131.

[10] P. Kanerva, "Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors," *Cogn. Comput.*, vol. 1, pp. 139–159, Jun. 2009.

[11] H. Amrouch et al., "Brain-inspired hyperdimensional computing for ultra-efficient edge AI," in *Proc. Int. Conf. Hardw./Softw. Codesign Syst. Synth. (CODES+ISSS)*, 2022, pp. 25–34.

[12] 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, 2022.

[13] D. Liang, J. Shiomi, N. Miura, and H. Awano, "DistriHD: A memory efficient distributed binary hyperdimensional computing architecture for image classification," in *Proc. IEEE 27th Asia South Pac. Design Autom. Conf. (ASP-DAC)*, 2022, pp. 43–49.

[14] L. Ge and K. K. Parhi, "Classification using hyperdimensional computing: A review," *IEEE Circuits Syst. Mag.*, vol. 20, no. 2, pp. 30–47, Jun. 2020.

[15] M. Imani, J. Morris, J. Messerly, H. Shu, Y. Deng, and T. Rosing, "BRIC: Locality-based encoding for energy-efficient brain-inspired hyperdimensional computing," in *Proc. 56th ACM/IEEE Design Autom. Conf. (DAC)*, 2019, pp. 1–6.

[16] M. Imani et al., "A framework for collaborative learning in secure high-dimensional space," in *Proc. IEEE 12th Int. Conf. Cloud Comput. (CLOUD)*, 2019, pp. 435–446.

[17] B. Khaleghi, H. Xu, J. Morris, and T. Š. Rosing, "tiny-HD: Ultra-efficient hyperdimensional computing engine for IoT applications," in *Proc. IEEE/ACM Design Autom. Test Eur. Conf. (DATE)*, 2021, pp. 408–413.

[18] A. Hernandez-Cane, 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)*, 2021, pp. 56–61.

[19] M. Imani, S. Salamat, B. Khaleghi, M. Samragh, F. Koushanfar, and T. Rosing, "SparseHD: Algorithm-hardware co-optimization for efficient high-dimensional computing," in *Proc. IEEE 27th Annu. Int. Symp. Field-Program. Custom Comput. Mach. (FCCM)*, 2019, pp. 190–198.

[20] M. Imani, C. Huang, D. Kong, and T. Rosing, "Hierarchical hyperdimensional computing for energy efficient classification," in *Proc. 55th ACM/ESDA/IEEE Design Autom. Conf. (DAC)*, 2018, pp. 1–6.

[21] 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*, 2016, pp. 64–69.

[22] Y. Kim, M. Imani, and T. S. Rosing, "Efficient human activity recognition using hyperdimensional computing," in *Proc. 8th Int. Conf. Internet Things*, 2018, pp. 1–6.

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

[24] M. Imani, D. Kong, A. Rahimi, and T. Rosing, "VoiceHD: Hyperdimensional computing for efficient speech recognition," in *Proc. IEEE Int. Conf. Rebooting Comput. (ICRC)*, 2017, pp. 1–8.

[25] M. Imani, S. Pampana, S. Gupta, M. Zhou, Y. Kim, and T. Rosing, "DUAL: Acceleration of clustering algorithms using digital-based processing in-memory," in *Proc. 53rd Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, 2020, pp. 356–371.

[26] D. Liang, M. Hashimoto, and H. Awano, "BloomCA: A memory efficient reservoir computing hardware implementation using cellular automata and ensemble bloom filter," in *Proc. IEEE/ACM Design Autom. Test Eur. Conf. (DATE)*, 2021, pp. 587–590.
[27] T. Clanuwat, M. Bober-Irizar, A. Kitamoto, A. Lamb, K. Yamamoto, and D. Ha, "Deep learning for classical Japanese literature," 2018, *arXiv:1812.01718*.
[28] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-MNIST: A novel image dataset for benchmarking machine learning algorithms," 2017, *arXiv:1708.07747*.
[29] "The MPLab GENKI database." Accessed: Dec. 27, 2023. [Online]. Available: http://mplab.ucsd.edu
[30] R. Balasubramonian, A. Kahng, N. Muralimanohar, A. Shafiee, and V. Srinivas, "CACTI 7: New tools for interconnect exploration in innovative off-chip memories," *ACM Trans. Archit. Code Optim.*, vol. 14, pp. 1–25, Jun. 2017.

DEHUA LIANG received the B.E. degree in microelectronic science and engineering from Xian Jiaotong University, China, in 2018. He is currently pursuing the Ph.D. degree with the Department of Information Systems Engineering, Osaka University, Japan.

JUN SHIOMI (Senior Member, IEEE) received the B.E. degree in electrical and electronics engineering, the M.E. degree in Informatics, and the Ph.D. degree in Informatics from Kyoto University, Kyoto, Japan, in 2014, 2016, and 2017, respectively. From 2016 to 2017, he was a Research Fellow with the Japan Society for the Promotion of Science. From December 2017 to March 2021, he was an Assistant Professor with the Department of Communications and Computer Engineering, Graduate School of Informatics, Kyoto University. In April 2021, he joined Osaka University, where he is currently an Associate Professor with the Graduate School of Information Science and Technology. His research interests include modeling and computer-aided design for low power and low voltage system-on-chips.

NORIYUKI MIURA (Member, IEEE) received the B.S., M.S., and Ph.D. degrees in electrical engineering from Keio University, Yokohama, Japan. From 2005 to 2008, he was a JSPS Research Fellow and since 2007 has been an Assistant Professor with Keio University, where he developed wireless interconnect technology for 3-D integration. In 2012, he moved to Kobe University, Kobe, Japan, and became a Professor with Osaka University, Suita, Japan, in 2020. Also, he was concurrently appointed as a JST PRESTO Researcher, and currently working on hardware security/safety and next-generation heterogeneous computing systems. He was a recipient of the Top ISSCC Paper Contributors from 2004 to 2013, the IACR CHES Best Paper Award in 2014, the IEICE Suematsu Yasuharu Award in 2017, and the Marubun Research Encouragement Award in 2019. He is currently serving as a Technical Program Committee (TPC) Member for A-SSCC and Symposium on VLSI Circuits. He served as the TPC Vice Chair of 2015 A-SSCC.

HIROMITSU AWANO (Member, IEEE) received the B.S., M.S., and Ph.D. degrees in informatics from Kyoto University in 2010, 2012, and 2016, respectively. He was with Hitachi, Ltd., Tokyo, Japan, in 2016, with the VLSI Design and Education Center, The University of Tokyo, Japan, from 2017 to 2018, and with the Graduate School of Information Science and Technology, Osaka University, Osaka, Japan, from 2019 to 2020. In 2020, he joined the Graduate School of Informatics, Kyoto University, Kyoto, Japan, where he is currently an Associate Professor. His research interests include CAD for VLSI design and hardware accelerator for machine learning. He was a Research Fellow with the Japan Society for the Promotion of Science, and a member of IEICE and IPSJ.