SearcHD: A Memory-Centric Hyperdimensional Computing with Stochastic Training

Mohsen Imani, Xunzhao Yin, John Messerly, Saransh Gupta, *Student Member, IEEE*, Michael Niemier, *Senior Member, IEEE*, Xiaobo Sharon Hu, *Fellow, IEEE,* and Tajana Rosing, *Fellow, IEEE*

***Abstract*—Brain-inspired HyperDimensional (HD) computing emulates cognitive tasks by computing with long binary vectors–aka hypervectors–as opposed to computing with numbers. However, we observed that in order to provide acceptable classification accuracy on practical applications, HD algorithms need to be trained and tested on non-binary hypervectors. In this paper, we propose SearcHD, a fully binarized HD computing algorithm with a fully binary training. SearcHD maps every data points to a high-dimensional space with binary elements. Instead of training an HD model with non-binary elements, SearcHD implements a full binary training method which generates multiple binary hypervectors for each class. We also use the analog characteristic of non-volatile memories (NVMs) to perform all encoding, training, and inference computations in memory. We evaluate the efficiency and accuracy of SearcHD on a wide range of classification applications. Our evaluation shows that SearcHD can provide on average 31.1× higher energy efficiency and 12.8× faster training as compared to the state-of-the-art HD computing algorithms.**

***Index Terms*—Brain-inspired computing, Hyperdimensional computing, Processing in-memory**

I. Introduction

The existing learning algorithms have been shown to be effective for many different tasks, e.g., object tracking [1], speech recognition [2], [3], image classification [4], [5], etc. For instance, Deep Neural Networks (DNNs) have shown great potential to be used for complicated classification problems. DNN architectures such as AlexNet [4] and GoogleNet [6] provide high classification accuracy for complex image classification tasks, e.g., ImageNet dataset [7]. However, the computational complexity and memory requirement of DNNs makes them inefficient for a broad variety of real-life (embedded) applications where the device resources and power budget is limited.

Brain-inspired computing models in conjunction with recent advancements in memory technologies have opened new avenues for efficient execution of a wide variety of cognitive tasks on nano-scale fabrics [8]–[11]. Hyperdimensional (HD) computing is based on the understanding that brains compute with *patterns of neural activity* that are not readily associated with numbers [12]. HD computing builds upon a well-defined set of operations with random HD vectors and is extremely robust in the presence of hardware failures. HD computing offers a computational paradigm that can be easily applied to learning problems [12]–[19]. Its main differentiation from conventional computing system is that in HD computing, data is represented as approximate patterns, which can favorably scale for many learning applications.

HD computation runs in three steps: encoding, training, and inference [20].

- The encoding module maps input data into high-dimensional space using a set of randomly generated hypervectors.
- In training, a traditional HD algorithm combines the encoded hypervectors in order to generate a hypervector representing each class. The algorithm simply performs element-wise additions on the hypervectors which belong to the same class.
- In inference, an associative search checks the similarity of an encoded test hypervector with all trained class hypervectors and returns the class with the highest similarity score.

While HD computing can be implemented in conventional digital hardware, implementation on approximate hardware can yield substantial efficiency gains with minimal to zero loss in accuracy [21].

Processing in-memory (PIM) is a promising solution to accelerate HD computations running for memory-centric applications by enabling parallelism [22]–[30]. PIM performs some or all of a set of computation tasks (e.g., bit-wise or search computations) inside the memory without using any processing cores. Thus application performance may be accelerated significantly by avoiding the memory access bottleneck. In addition, PIM architectures enable analog-based computation in order to perform approximate but ultra-fast computation (i.e., existing PIM architectures perform computation with binary vectors stored in memory rows [31]). Past efforts have tried to accelerate HD computing via PIM. For example, work in [8], [32] designed in-memory hardware to accelerate the encoding module. Work in [21] designed a content-addressable memory which can perform the associative search operations for inference over binary hypervectors using a Hamming distance metric. However, the aforementioned accelerators can only work with binary vectors, which in turns only provide high classification accuracies on simpler problems, e.g., language recognition which uses small n-gram windows of size five to detect words in a language. In this work, we observed

D. Imani, J. Messerly, S.Gupta, and T. Rosing are with the Department of Computer Science and Engineering, University of California San Diego, La Jolla, CA, 92093.
E-mail: {moimani, jmesserl, sgupta, tajana}@ucsd.edu

X. Yin, M. Niemier and X. S. Hu are with the Department of Computer Science and Engineering, University of Notre Dame, Notre Dame, IN, 46556.
E-mail:{xyin1, mniemier, shu}@nd.edu

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2019.2952544, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

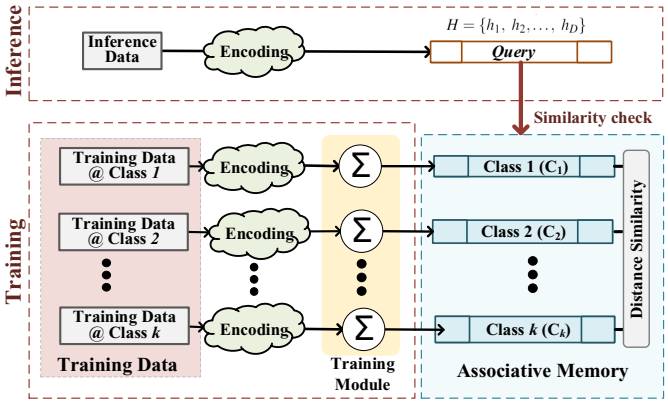


Fig. 1. Overview of HD computing in performing the classification task.

that acceptable classification accuracy can be achieved using non-binary encoded hypervectors, non-binary training, and associative search on a non-binary model using metrics such as cosine similarity. This hinders the implementation of many steps of the existing HD computing algorithms using in-memory operations.

In order to fully exploit the advantage of in-memory architecture, we propose SearcHD, a fully binary HD computing algorithm with probability-based training. SearcHD maps every data point to high-dimensional space with binary elements and then it assigns multiple vectors representing each class. Instead of performing addition, SearcHD performs binary training by changing each class hypervector depending on how well it matches with a class that it belongs to. Unlike most recent learning algorithms, e.g., neural networks, SearcHD supports a single-pass training, where it trains a model by one time passing through a training dataset. The inference step is performed by using a Hamming distance similarity check of a binary query with all pre-stored class hypervectors. All HD computing blocks including encoding, training, and inference are implemented fully in memory without using any non-binary operation. SearcHD exploits the analog characteristic of ReRAMs to perform the encoding functionalities such as XOR and majority functions, and training/inference functionalities such as the associative search on ReRAMs.

We have tested the accuracy and efficiency of the proposed SearcHD on four practical classification applications. Our evaluation shows that SearcHD can provide on average 31.1× higher energy efficiency and 12.8× faster training as compared to the state-of-the-art HD computing algorithms. In addition, during inference, SearcHD can achieve 178.7× higher energy efficiency and 14.1× faster computation while providing 6.5% higher classification accuracy than state-of-the-art HD computing algorithms.

II. Background and Motivation

Hyperdimensional computation is a novel computational paradigm inspired by how the brain represents data. HD computing has previously shown to address energy bounds which plague deterministic computing [12], [33]. HD computing replaces the conventional computing approach with *patterns of neural activity* that are not readily associated with numbers. Due to the large size of brain circuits, this neurons pattern can be represented using vectors in thousands of dimensions, which are called hypervectors. Hypervectors are holographic and (pseudo) random with i.i.d. components. Each hypervector stores the information across all its components, where no component has more responsibility to store any piece of information than another. This makes HD computing extremely robust against failures. HD computing supports a well-defined set of operations such as *binding* that forms a new hypervector which associates two hypervectors, and *bundling* that combines several hypervectors into a single composite hypervector. Reasoning in HD computing is based on the similarity between the hypervectors.

Prior work applied HD computing to a wide range of applications including: analogy-based reasoning [34], latent semantic analysis [35], language recognition [36], [37], text classification [38], gesture recognition [39], prediction from multimodal sensor fusion [14], [40], robotics [41], and speech recognition [15], [42]. Figure 1 shows an overview of how HD computing performs a classification task. The first step in HD computing is to map (encode) raw data into a high-dimensional space. Various encoding methods have been proposed to handle different data types such as time series, text-like data, and feature vectors [13], [15]–[17], [20], [42]. Regardless of the data type, the encoded data is represented with a D dimensional vector ($H \in \mathbb{N}^D$) [15], [42]. Training is performed by computing the element-wise sum of all hypervectors corresponding to the same class ($\{C_1, \ldots, C_K\}, C_i \in \mathbb{N}^D$), as shown in Figure 1. For example, in an application with k classes, the i^{th} class hypervector can be computed as:

$$C_i = \sum_{\forall j \in class_i} H_j$$

This training operation involves many integer (non-binary) additions, which makes the HD computation costly. During inference, we simply compare a similarity score between an encoded *query hypervector* and each class hypervector, returning the most similar class. Prior work has typically used the cosine similarity (inner product) which involves a large number of non-binary additions and multiplications. For example, for an application with k classes, this similarity check involves $k \times D$ multiplication and addition operations, where the hypervector dimension is D, commonly 10,000.

In terms of hardware acceleration, prior work tried to binarize the HD trained model in order to simplify HD computing inference [20], [42]. Thanks to the inherent memory-centric operations of HD computing, several works design in-memory architectures to accelerate HD computing [8], [21], [32]. The work in [8] fabricated a 3D VRRAM/CMOS to support the central operations of HD computing on 4-layer 3D VRRAM/FinFET. Work in [21] designed three popular digital, resistive and analog associative memories in order to accelerate Hamming distance similarity in HD computing inference. The work in [32] exploited carbon nanotube FETs and resistive memory to design an HD computing algorithm for text clas-

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2019.2952544, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

sification. However, these methods only support Hamming distance similarity between binarized hypervectors. In this paper, we observe that HD computing algorithms require the use of integer values and *cosine* similarity metrics in order to provide acceptable accuracies on practical classification problems.

Table I shows the classification accuracy and the inference efficiency of HD computing on four practical applications (large feature size) when using binary and non-binary models. All efficiency results are reported for running the applications on digital ASIC hardware [15], [20]. Our evaluation shows that HD computing with the binary model has 4% lower classification accuracy than the non-binary model. However, in terms of efficiency, HD computing with the binary model can achieve on average 6.1× faster computation than the non-binary model. In addition, HD computing with the binary model can use Hamming distance for similarity check of a query and class hypervectors which can be accelerated in a content addressable memory (CAM) [21], [47]. Our evaluation shows that such analog design can further speedup the inference performance by 6.9× as compared to digital design. This motivates us to design a novel HD computing algorithm which can achieve the efficiency of a binary model as well as the accuracy of the non-binary model.

III. SearcHD: A Fully Binary HD Computing

We propose SearcHD, a fully binary HD computing algorithm, which can perform all HD computing operations, i.e., encoding, training, and inference, using binary operations. In the rest of the section, we explain the details of the proposed approach. We first explain the functionality of the encoding, training, and inference modules, and then illustrate how module functionality can be supported in hardware.

A. SearcHD Encoding

There are different types of encoding methods to map data points into an HD space. For a general form of feature vectors, there are two popular encoding approaches; (i) record-based and (ii) Ngram-based encoding [15]. Although SearcHD functionality is independent of the encoding module, here we use a record-based encoding which is more hardware friendly and only involves bitwise operations as shown in Figure 2a. This encoding maps any feature vector $F = \{f_1,\ f_2, \ldots,\ f_n\}$ with n features ($f_i \in \mathbb{N}$), into $H = \{h_1,\ h_2, \ldots,\ h_D\}$ with D dimensions ($h_i \in \{0,1\}$) [15], [48]. This encoding finds the minimum and maximum feature values and quantizes that range linearly into m levels. Then, it assigns a random binary hypervector with D dimensions to each of the quantized level $\{L_1, \ldots, L_m\}$. The level hypervectors need to have correlation, such that the neighbor levels are assigned to similar hypervectors. For example, we generate the first level hypervector, L_1, by sampling uniformly at random from 0 or 1 values. The next level hypervectors are created by flipping D/m random bits of the previous level. As a result, the level hypervectors have similar values if the corresponding original data are closer, while L_1 and L_m will be nearly orthogonal. The orthogonality between the bipolar/binary hypervectors defines when two vectors have exactly 50% similar bits. This results in a zero cosine similarity between the orthogonal vectors.

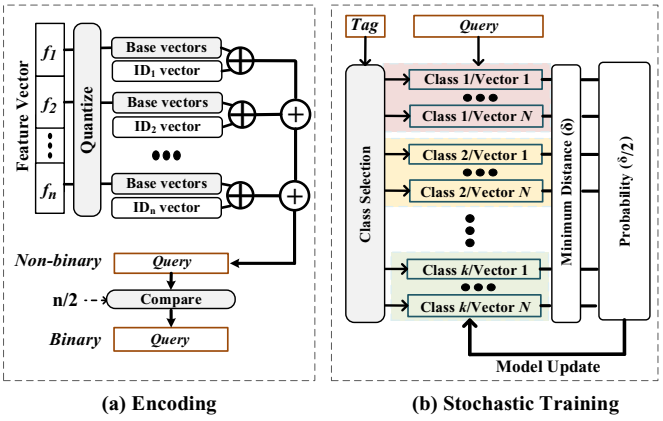


Fig. 2. Overview of SearcHD encoding and stochastic training.

Similarly, the encoding module assigns a random binary hypervector to each existing feature index, $\{ID_1, \ldots, ID_n\}$, where $ID \in \{0,1\}^D$. The encoding linearly combines the feature values over different indices:

$$H \ = \ ID_1 \oplus \overline{L}_1 \ + \ ID_2 \oplus \ \overline{L}_2 + \ldots \ + \ ID_n \oplus \overline{L}_n.$$

where H is the non-binary encoded hypervector, $\oplus$ is XOR operation, and $\overline{L}_i \in \{L_1, \ldots, L_m\}$ is the binary hypervector corresponding to the i-th feature of vector F. In this encoding, IDs preserve the position of each feature value in a combined set. SearcHD encodes this hypervector by passing each dimension through a *majority* function. This approach compares each dimension with a threshold value, where a threshold value is equal to half of the number of features ($THR = n/2$).

B. Binary Stochastic Training

In HD computing, when two data hypervectors are randomly generated, the probability that their hypervectors are orthogonal to each other is high. In training, hypervectors of data in the same class are made appropriately more or less similar to each other. In more recent HD computing algorithms [15], [20], [39], training consists of additions of the encoded hypervectors, thus requiring a large number of arithmetic operations to generate non-binary class hypervectors. Our proposed SearcHD is a framework for binarization of the HD computing algorithm during both training and inference. SearcHD removes the addition operation from training by exploiting *bitwise substitution* which trains a model by stochastically sharing the query hypervectors elements with each class hypervector. Since HD computing with a binary model provides low classification accuracy, SearcHD exploits *vector quantization* to represent an HD model using multiple vectors per class. This enables SearcHD to store more information in each class while keeping the model as binary vectors.

1) SearcHD Bitwise Substitution: SearcHD removes all arithmetic operations from training by replacing addition with bitwise substitution. Assume A and B are two randomly

TABLE I
CLASSIFICATION ACCURACY AND EFFICIENCY OF HD COMPUTING USING BINARY AND NON-BINARY MODELS.

Applications	Accuracy		Inference Execution (ms)		
	Non-Binary	*Binary*	*Non-Binary*	*Binary*	*Analog* [21]
Speech Recognition [43]	83.4%	78.6%	3240.5	538.4	74.1
Cardiotocograms [44]	78.7%	73.5%	791.0	131.8	18.2
Activity Recognition [45]	82.3%	79.2%	2025.2	337.7	50.9
Security Prediction [46]	94.0%	91.4%	494.3	272.9	13.2

generated vectors. In order to bring vector A closer to vector B, a random (typically small) subset of vector B's indices is forced onto vector A by setting those indices in vector A to match the bits in vector B. Therefore, the Hamming distance between vector A and B is made smaller through partial cloning. When vector A and B are already similar, then indices selected probably contain the same bits, and thus the information in A does not change. This operation is blind since we do not search for indices where A and B differ, and then "fix" those indices. Indices are chosen randomly and independently of whatever is in vector A or vector B. In addition, the operation is one directional. Only the bits in vector A are transformed to match those in vector B, while the bits in vector B stay the same. In this sense, A inherits an arbitrary section of vector B. We call vector A the *binary accumulator* and vector B the *operand*. We refer to this process as *bitwise substitution*.

2) SearcHD Vector Quantization: Here, we present our fully binary stochastic training approach, which enables the entire HD training process to be performed in the binary domain. Similar to traditional HD computing algorithms, SearcHD trains a model by combining the encoded training hypervectors. As we explained in Section II, HD computing using binary model results in very low classification accuracy. In addition, moving to the non-binary domain makes HD computing significantly more costly and inefficient. In this work, we propose vector quantization We exploit multiple vectors to represent each class in the training of SearcHD. The training keeps distinct information of each class in separated hypervectors, resulting in the learning of a more complex model when using multiple vectors per class. For each class, we generate N models (where N is generally between 4 and 64). Below we explain the details of the proposed algorithm:

- Initialize the N model vectors to a class by randomly sampling from the encoded training hypervector of that class as shown in Figure 2b. For an application with k classes, the approach needs to store $N \times k$ binary hypervectors as the HD model. For example, we can represent the wi^{th} class using N initial binary hypervectors $\{C_1^i, C_2^i, \cdots, C_N^i\}$, where $C^i \in \{0,1\}^D$
- The training in HD computing starts by checking the similarity of each encoded data point (training dataset) to the initial model. The similarity check only happens between the encoded data and N class hypervectors corresponding to that label. For each piece of training data Q in a class, find the model with the lowest Hamming distance and update the model using bitwise substitution (explained in Section III-B1). For example, in the i^{th} class, if C_k^i is selected as a class with the highest similarity, we can update the model using:

$$C_k^i = C_k^i \ (+) \ Q$$

In the above equation, (+) is the bitwise substitution operation, Q is the *operand*, and C_k^i is the binary accumulator. This algorithm helps reduce the memory access overhead introduced by using bitwise substitution. This approach accumulates training data more intelligently: given the choice of adding an incoming piece of data to one of N model vectors, we can select the model with the lowest Hamming distance to ensure that we do not needlessly encode information in our models.

3) SearcHD Training Process: Binary substitution updates each dimension of the selected class stochastically with $p = \alpha \times (1-\delta)$ probability, where δ is a similarity between the query and the class hypervector and α is a learning rate. In other words, with flip probability p, each element of the selected class hypervector will be replaced with the elements of the query hypervector. α is the learning rate ($0 < \alpha$) which determines how frequently the model needs to be updated during the training. Using a small learning rate is conservative, as the model will have minor changes during the training. A larger learning rate will result in a major change to a model after each iteration, resulting in a higher probability of divergence. We explore the impact of learning rate on the classification accuracy in Section V-B

C. Inference

After updating the model on the entire training dataset, SearcHD uses the trained model for the rest of the classification during inference. The classification checks the similarity of each encoded test data vector to all class hypervectors. In other words, a query hypervector is compared with all $N \times k$ class hyprevectors. Finally, a query identifies a class with the maximum Hamming distance similarity with the query data.

IV. IN-MEMORY CLASSIFICATION IN HD SPACE

SearcHD requires bitwise computations over hypervectors in both training and inference modes. These operations are fast and efficient when compared to floating point operations used by neural networks or other classification algorithms [8], [21]. This enables HD computing to be trained and tested on light-weight embedded devices. However, as traditional CPU/GPU cores have not been designed to efficiently perform bitwise operations over long vectors, we provide a custom hardware realization of SearcHD. Here we show how to use the analog

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2019.2952544, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

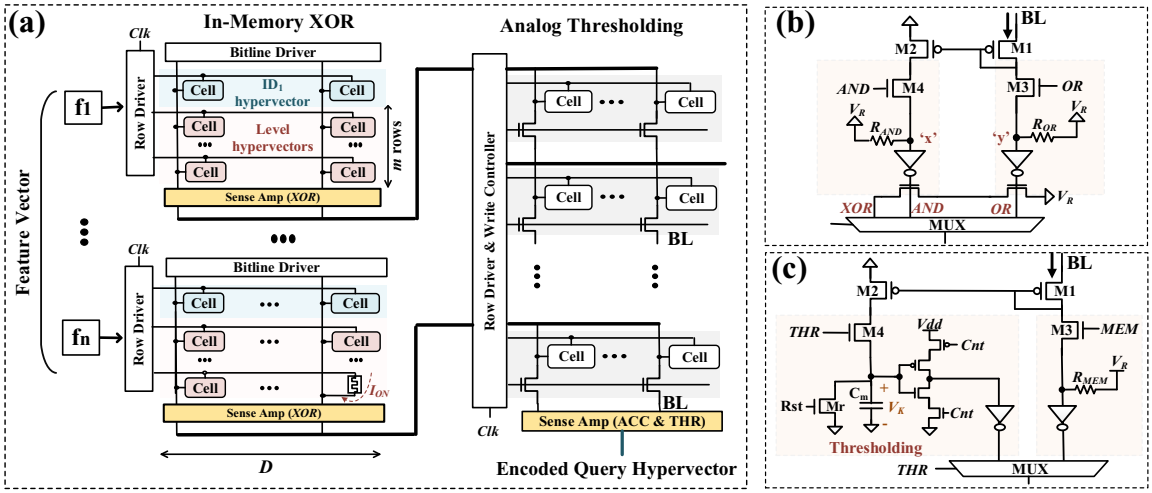


Fig. 3. (a) In-memory implementation of SearcHD encoding module. (b) The sense amplifier supporting bitwise XOR operation, and (c) the sense amplifier supporting majority functionality on the XOR results.

characteristics of ReRAMs, in particular ReRAM devices, to process all HD computing functionality within a memory.

A. Overview

HD computing operations can be supported using two main encoding and associative search blocks. In Section IV-B, we explain the details of the in-memory implementation of the encoding module.

During training, SearcHD requires only a single pass over the training set. For each class, SearcHD first randomly selects N data points from the training dataset as representative class hypervectors. Then, SearcHD uses CAM blocks to check the similarity of each encoded hypervector (from the training dataset) with the class hypervectors. Depending on the tag of input data, SearcHD only needs to perform the similarity check on N hypervectors of the same class as input data. For each training sample, we find a hypervector in a class which has the highest similarity with the encoded hypervector using a memory block which supports the nearest Hamming distance search. Then, we update the class hypervector depending on how well/close it is matched with the query hypervector (δ). To implement this, we exploit an analog characteristic of ReRAMs to identify how well a query data is matched with the most similar class. We exploit the feature of ReRAM devices to generate a random stochastic sequence with the desired probability. This sequence needs to have a similar number of zeros as the difference between a query and the class hypervector. Finally, we update the elements of a selected class by performing an in-memory XNOR operation between (i) the class and (ii) the stochastically generated hypervector. In the following section, we explain how SearcHD can support all of these operations in an analog fashion.

During inference, we employ the same CAM block used in training to implement the nearest Hamming distance search between a query and all class hypervectors. For an application with k classes and N class hypervectors, SearcHD requires similarity comparisons between the query and $k \times N$ stored class hypervectors. The hardware introduced in Section IV-C, explains the details of in-memory search implementation.

B. In-Memory Encoding

The encoder, shown in Figure 3a, implements bitwise XOR operations between hypervectors P and L over different features, and thresholds the results. To support this functionality, our analog design assigns a small size crossbar memory ($m+1$ rows with D dimensions) to each input feature, where the crossbar memory stores the corresponding position hypervector (ID) along with all m possible level hypervectors that each feature can take (m is the number of level hypervectors, as defined in Section III). The results of all XOR operations are written to another crossbar memory. Next, the memory that stores the XOR results perform the bitwise majority operation on the entire memory. In conventional crossbar memory, the write in the majority block needs to perform serially over different features. However, in this work, we use switches (shown in Figure 3a) that enables parallel write in the majority block. These switches portioned memory rows into a separated block, enabling all XOR results to be written in majority block independently. We accomplish this operation by designing a sense amplifier for the crossbar memory that can detect whether the number of 1's is above a certain threshold (THR). This *Majority* function does not involve actual counting but uses the analog characteristics of ReRAM devices, making it a significantly faster and more efficient design.

In-Memory XOR: To enable an XOR operation as required by the encoding module, the row driver must activate the line corresponding to the position hypervector (ID shown in Figure 3a). Depending on the feature value, the row driver activates one more row in the crossbar memory which corresponds to the feature value. Our analog design supports bitwise XOR operations inside the crossbar memory among two activated rows. This design enables in-memory XOR operations by making a small modification to the sense amplifier of the crossbar memory, as shown in Figure 3b. We place a

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2019.2952544, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

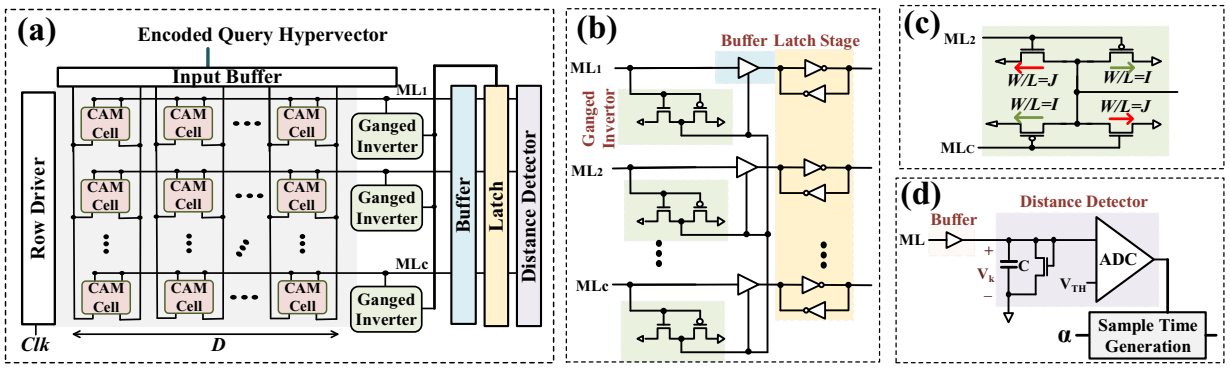


Fig. 4. (a) CAM-based associative memory. (b) The structure of the CAM sense amplifier, and (c) the ganged circuit and (d) the distance detector circuit.

modified sense amplifier at the tail of each vertical bitline (*BL*). The *BL* current passes through the R_{OR} and R_{AND}, and changes the voltage in node 'x' and 'y'. A voltage larger than a threshold in node 'x' and 'y' results in inverting the output values of the inverters, realizing the AND and OR operations. We use the combination of AND and OR operations to generate XOR. In our design, R_{OR}, R_{AND}, and V_R are tuned to ensure the correct functionality of the design considering process variations. It should be noted that the same XOR functionality could be implemented using a series of MAGIC NOR operation introduced by S. Kvatinsky [49]. The advantage of this approach is that we do not need to make any changes to the sense amplifier. However, the clock cycle of MAGIC NOR is at the order of 1ns, while the proposed approach computes XOR in less than 300ps.

In-Memory Majority: Figure 3c shows the sense amplifier designed to implement the majority function. To evaluate this function, a row driver activates all rows of the crossbar memory. Any cell with low resistance injects current into the corresponding vertical *BL*. The number of 0s in each column determines the amount of current in the *BL*. The charging rate of the capacitor C_m in the proposed sense amplifier depends on the number of zeroes in each column. The sense amplifier samples the capacitor voltage at a *specific time*, which intuitively is the same as comparing the *BL* current with a $THR = n/2$ value. Since the charging current is constant, the voltage grows linearly with time, thus the "specific time" can be equal to twice the time where C_m is charged by the maximum current. Our design can use different pre-determined THR values in order to tune the level of *thresholding* for applications with different feature sizes.

C. In-Memory Associative Search

The goal of this search operation is to find a class with the highest similarity. We employ crossbar CAMs, which can search the nearest Hamming distance vector with respect to the stored class hypervectors. Traditionally, CAMs are only designed to search for an exact match and cannot look for a row with the smallest Hamming distance.

Figure 4a shows an architectural schematic of a conventional CAM. A search operation in CAM starts with pre-charging all CAM match-lines (*MLs*). An input data vector is applied to a CAM after passing through an input buffer. The goal of the buffer is to increase the driving strength of the input data and distribute the input data across the entire memory at approximately the same time. Finally, each CAM row is compared with the input data. Conventional CAM can detect a row that contains an exact matching, i.e., where all bits of the row exactly match with the bits in the input data.

In this work, we exploit the analog characteristics of CAM blocks to detect the row that has the minimum Hamming distance with the query vector. Each CAM row with stored bits that are different from the provided input will discharge the associated *ML*. The rate of *ML* discharging depends on the number of mismatch bits in each CAM row. A row with the nearest Hamming distance to the input data is the one with the lowest *ML* discharging current, resulting in the longest discharging time. To find this row, we need to keep track of all other *MLs* and determine when only a single *ML* is still discharging.

Figure 4b shows the general structure of the proposed CAM sense amplifier. We implement nearest Hamming distance search functionality by detecting the CAM row (most closely matched line) which discharges last. This is realized with three main blocks: (i) detector circuitry which samples the voltage of all *MLs* and detects the *ML* with the slowest discharge rate, (ii) a buffer stage which delays the *ML* voltage propagation to the output node, and (iii) a latch block which samples buffer output when the detector circuit detects that all *MLs* are discharged. The last edge detection can be easily implemented by NORing the outputs of all matched lines, which is set when all *MLs* are discharged to zero. However, a conventional implementation of NOR leads to a huge number of transistors and increased latency as the number of inputs rises beyond 3.

Here, we propose the use of a *Ganged* CMOS-based [50] NOR circuit, which not only provides faster results but can also support larger fan-in with acceptable noise margin. Figures 3b and c show the circuit consisting of skewed inverters with their outputs shorted together. The behavior of the circuit is defined by the ratio $r = I/J$, where I and J are the sizes of the pull-up and pull down transistors, respectively. For $r > q$, the inverter is highly skewed and has a stronger pull-up, while for $r < q$, it is lowly skewed and has a stronger pull-down. (q is a technology dependent parameter.) If r is sufficiently small,

outputs of multiple skewed inverters can be shorted together to implement the NOR operation. Figures 4b and c show that the output of the circuit is set only when all the nMOS devices are off, i.e. all *ML*s are zero. The output of ganged NOR circuit controls the latch. The buffer stage used in the sense circuit adds delay to the *ML* voltage. This delay should be sufficiently large to ensure that we can latch on the last falling *ML*, when the ganged logic senses that all *ML*s have fallen to zero.

D. In-Memory Distance Detector

In training, SearcHD uses the proposed CAM block to find a hypervector which has the highest similarity with a query data. Then, SearcHD needs to update the selected class hypervector with the probability that is proportional to how well a query is matched with the class. After finding a class hypervector with the highest similarity, SearcHD performs the search operation on the selected row. This search operation finds how closely the selected row matches with the query data. This can be sensed by the distance detector circuit shown in Figure 4d. Our analog implementation transfers the discharging current of a CAM row into a voltage (V_k) and compares it with a reference voltage (V_{TH}). The reference voltage is the minimum voltage that V_K can take when all query dimensions of a query hypervector match with the class hypervector.

E. In-Memory Random Generation & Model Update

Depending on the distance similarity difference betweem a query and the class hypervector, SearcHD generates a random sequence of a bit stream which has a proportional number of 1s. For example, for a query with δ similarity between the query and the class hypervector, SearcHD generates a random sequence with $p = \alpha \times (1-\delta)$ probability of '1' bits. During training, SearcHD selects the class hypervector with the minimum Hamming distance from the query data, and updates the selected class hypervector by bitwise substitution of a query and the class hypervector. This bitwise substitution is performed stochastically on random $p \times D$ of the class dimensions. This requires generating a random number with a specific probability.

ReRAM switching is a stochastic process, thus the write operation in a memristor device happens with a probability which follows a Poisson distribution [51], [52]. This probability depends on several factors such as programming voltage and write pulse time. For a given programming voltage, we can define the switching probability as:

$$P(t) = 1 - e^{-t/\tau} \qquad \tau(V) = \tau_0 e^{-V/V_0}$$

where τ is the characterized switching time that depends on the programming voltage, V, and τ_0 and V_0 are the fitting parameters. To ensure memristor switching with high probability, the pulse width should be long enough. For example, using $t = \tau$, the switching probability is as low as $P(t = \tau) = 63\%$, while using $t = 10\tau$ increases this probability to $P(t = 10\tau)$ =99.995%. We exploit the non-deterministic ReRAM switching property to generate random numbers [53], [54]. Depending on the applied voltage, a pulse time is assigned to set the device switching probability to the desired percentage. For example, to generate numbers with 50% probability, the pulse time (α) has been set to ensure $p(t = \alpha\tau) = 50\%$.

Assume a class hypervector with D dimensions, $C^i = \{c_1, c_2, \cdots, c_D\}$. Random generation creates a bit sequence with D dimensions, $R = \{r_1, r_2, \cdots, r_D\}$ but with p probability of bits to be '1'. We update the selected class hypervector in two steps: **(i)** we read the random hypervector R using a memory sense amplifier to select the bits for the bitwise substitution operation. Then, we activate the row of a selected class hypervector and apply R as a bitline buffer to reset corresponding class elements where R has '1' values there. **(ii)**, SearcHD reads the query hypervector (Q) and calculates the AND operation of the query and R hypervector. Our design uses the result of the AND operation as a bitline buffer in order to set the class elements in all dimensions where the bitline buffer has a '1' value. This is equivalent to injecting the query elements into a class hypervector in all dimensions where R has non-zero values.

V. Evaluation

In this section, we test the functionality of SearcHD in both software and hardware implementations. We first discuss the impact of learning rate and class configurations on SearcHD classification accuracy. We then compare the energy efficiency and performance of SearcHD with baseline HD computing during training and inference. We finally discuss the accuracy-efficiency tradeoff with respect to hypervector dimensions.

A. Experimental Setup

We test the functionality of SearcHD in both software and hardware implementations. In software, we verify SearcHD training and inference functionalities by a C++ implementation of the stochastic algorithm on an Intel Core i7 7600 CPU. For the hardware implementation, we have designed a cycle accurate simulator which emulates HD computing functionality. Our simulator pre-stores the randomly generated level and position hypervectors in memory and performs the training and inference operations fully in the proposed in-memory architecture. We extract the circuit level characteristic of the hardware design from simulations based on a 45nm CMOS process technology [55] using the Hewlett Simulation Program with Integrated Circuit Emphasis (*HSPICE*) simulator. We use the VTEAM ReRAM model [56] for our memory. The model parameters of the device, as listed in Table II, are chosen to produce switching delay of 1ns, a voltage pulse of 1V and 2V for RESET and SET operations in order to fit practical devices [49]. The energy of set and reset operations are $E_{set} = 23.8fJ$ and $E_{reset} = 0.32fJ$, respectively. The functionality of all the circuits has been validated considering 10% process variations on threshold voltage, transistor sizes, and ReRAM OFF/ON resistance using 5000 Monte Carlo simulations. Table III lists the design parameters including the transistor sizes, and AND/OR resistance values.

We test SearcHD accuracy and energy/performance efficiency on four practical classification applications. Table IV summarizes the configurations with various evaluation datasets.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2019.2952544, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

TABLE II
VTEAM MODEL PARAMETERS FOR MEMRISTOR

$\mathbf{k_{on}}$	$-216.2m/sec$	$\mathbf{V_{T,ON}}$	$-1.5V$	$\mathbf{x_{off}}$	$3nm$
$\mathbf{k_{off}}$	$0.091m/sec$	$\mathbf{V_{T,OFF}}$	$0.3V$	$\mathbf{R_{ON}}$	$10k\Omega$
$\alpha_{on}, \alpha_{off}$	4	$\mathbf{x_{on}}$	0	$\mathbf{R_{OFF}}$	$10M\Omega$

TABLE III
CIRCUIT PARAMETERS

Transistor Size (W/L)					*Resistance Values* (Ω)			Voltage
M1	*M2*	*M3*	M4	*Mr*	*AND*	*OR*	*MEM*	V_R
2	4	2	4	1	1.2k	5k	5k	1V

TABLE IV
DATASETS (n: FEATURE SIZE, k: NUMBER OF CLASSES).

	n	K	Train Size	Test Size	Description
ISOLET	617	26	6,238	1,559	Speech recognition [43]
FACE	608	2	522,441	2,494	Face recognition [57]
UCIHAR	561	12	6,213	1,554	Activity recognition [45]
IOT	115	2	40,000	2,494	IoT Botnet detection [46]

TABLE V
THE IMPACT OF LEARNING RATE ON SEARCHD CLASSIFICATION ACCURACY

α	**0.1**	**0.5**	**1**	**2**	**3**
ISOLET	83.6%	85.2%	85.2%	82.9%	81.4%
FACE	88.4%	90.1%	90.2%	89.5%	88.7%
UCIHAR	90.2%	91.1%	91.3%	90.8%	90.0%
IOT	98.5%	99.7%	99.9%	97.9%	96.8%

B. SearcHD and Learning Rate

Table V shows the impact of the learning rate α on SearcHD classification accuracy. Our evaluation shows that using a very small learning rate reduces the capability of a model to learn since each new data can only have a minor impact on the model update. Larger learning rates result in more substantial changes to a model, which can result in possible divergence. In other words, large α values indicate that there is a higher chance that the latest training data point will change the model, but it does not preserve the changes that earlier training data made on the model. In this work, our evaluation shows that using α values of 1-2 provide the maximum accuracy for all tested datasets.

C. SearcHD Accuracy in Different Configurations

Figure 5 shows the impact of the number of hypervectors per each class N on SearcHD classification accuracy in comparison with other approaches. State-of-the-art HD computing approaches use a single hypervector representing each class. As the figure shows, for all applications, increasing the number of hypervectors per class improves classification accuracy. For example, SearcHD using eight hypervectors per class (8/class) and 16 hypervectors per class (16/class) can achieve on average 9.2% and 12.7% higher classification accuracy, respectively, as compared to the case of using 1/class hypervector when running on four tested applications. However,

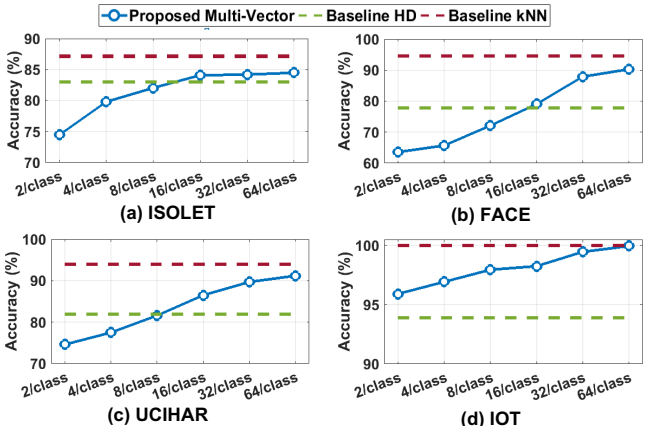


Fig. 5. Classification accuracy of SearcHD, kNN, and the baseline HD algorithms.

TABLE VI
MEMORY FOOTPRINT OF DIFFERENT ALGORITHMS (MB)

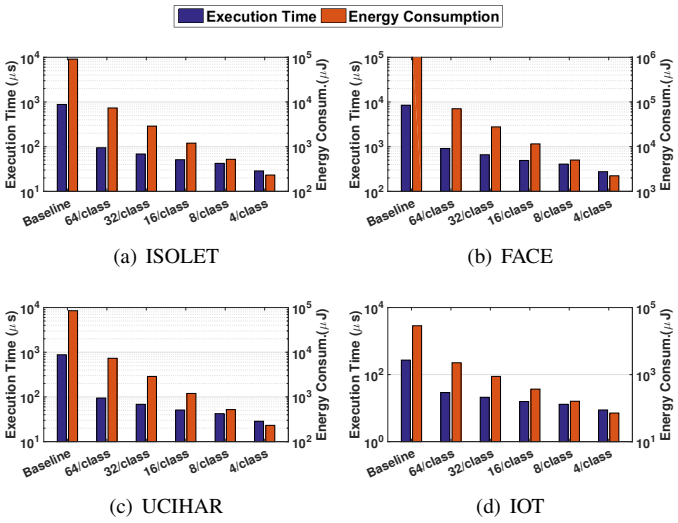
	kNN	Baseline HD	SearcHD				
			64/class	*32/class*	*16/class*	*8/class*	*4/class*
UISOLET	14.67	0.99	1.98	0.99	0.49	0.24	0.12
CARDIO	0.15	0.11	0.22	0.11	0.05	0.02	0.01
UCIHAR	13.29	0.45	0.91	0.45	0.22	0.11	0.05
IOT	17.54	0.07	0.15	0.07	0.04	0.02	0.01

SearcHD accuracy saturates when the number of hypervectors is larger than 32/class. In fact, 32/class is enough to get most common patterns in our datasets, thus adding new vectors cannot capture different patterns than the existing vectors in the class.

The red line in each graph shows the classification accuracy that a k-Nearest Neighbor (kNN) algorithm can achieve. kNN does not have a training mode. During Inference, kNN looks at the similarity of a data point with all other training data. However, kNN is computationally expensive and requires a large memory footprint. In contrast, SearcHD provides similar classification accuracy by performing classification on a trained model. Figure 5 also compares SearcHD classification accuracy with the best baseline HD computing algorithm using non-binary class hypervectors [15]. The baseline HD model is trained using non-binary encoded hypervectors. After the training, it uses a cosine similarity check for classification. Our evaluation shows that SearcHD with 32/class and 64/class provide 5.7% and 7.2% higher classification accuracy, respectively, as compared to the baseline HD computing with the non-binary model.

Table VI compares the memory footprint of SearcHD, kNN, and the baseline HD algorithm (non-binary model). As we expect, kNN has the highest memory requirement, by taking on average 11.4MB for each application. After that, SearcHD 32/class and the baseline HD algorithm require similar memory footprints, which are on average about 28.2× lower than kNN. SearcHD can further reduce the memory footprint by reducing the number of hypervectors per class. For example, SearcHD with 8/class configuration provides 117.1× and 4.1× lower memory than kNN and the baseline HD algorithm while providing similar classification accuracy.

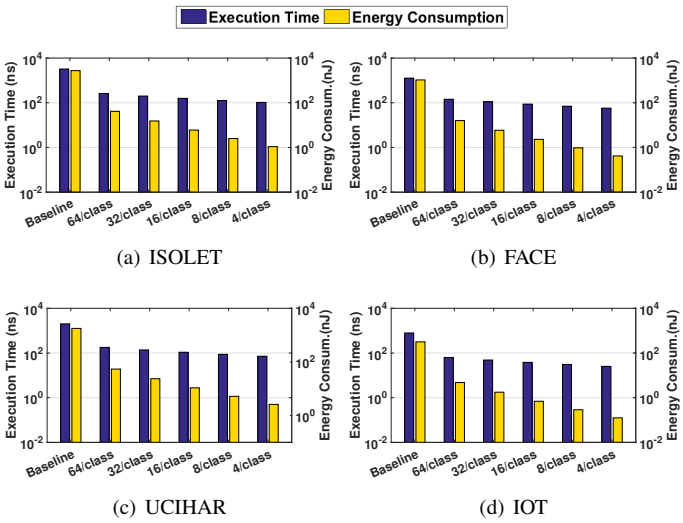
This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2019.2952544, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems



(a) ISOLET (b) FACE

(c) UCIHAR (d) IOT

Fig. 6. Training execution time and energy consumption of the baseline HD computing and SearcHD with different configurations.



(a) ISOLET (b) FACE

(c) UCIHAR (d) IOT

Fig. 7. Inference execution time and energy consumption of the baseline HD algorithm and SearcHD with different configurations.

D. Training Efficiency

Figure 6 compares the energy efficiency and performance of SearcHD training and the baseline HD computing algorithm. Regardless of whether binary or non-binary models are employed, the baseline HD computing approach has the same training cost. Baseline HD computing encodes data in the non-binary domain and then adds the input data in order to create a hypervector for each class. This operation cannot map into a crossbar memory architecture as the memory only supports the bit-wise operation. In contrast, SearcHD simplifies the training operation by eliminating all non-binary operations from HD training. In all reported results, we run SearcHD on the proposed in-memory architecture, while the baseline HD computing approach runs on optimized digital hardware proposed in [15] and [21]. Our evaluation shows that SearcHD with 64/class (32/class) configuration can achieve on average 12.2× and 9.3× (31.1× and 12.8×) higher energy efficiency and speedup as compared to the baseline HD computing algorithm.

E. Inference Efficiency

Figure 7 compares SearcHD and baseline HD computing efficiency during inference. The y-axis shows the energy consumptions and execution times of the baseline HD computing and SearcHD algorithm with the number of hypervectors per class ranging from 4 to 64. The baseline HD algorithm uses cosine as the similarity metric, while SearcHD uses Hamming distance and accelerates this computation via analog, in-memory hardware. Our evaluation shows that SearcHD with all configurations can provide significantly faster and more energy efficient computation as compared to the baseline HD algorithm. For example, SearcHD with 64/class (32/class) configuration can provide on average 66.2× and 10.8× (178.7× and 14.1×) energy efficiency and speedup as compared to a baseline HD algorithm, while providing 7.9% (6.5%) higher classification accuracy. The higher energy and performance efficiency of SearcHD comes from the in-memory capability in parallelizing the similarity check among different rows. In addition, the approximate search in analog memory eliminates slower digital-based counting operations.

In SearcHD, the computation cost grows with the number of hypervectors in a class. For example, SearcHD with 32/class configuration consumes 14.1× more energy and has 1.9× slower execution time as compared to SearcHD with 4/class configuration. In addition, we already observed that SearcHD accuracy saturates when using models with more than 32/class hypervector.

1) Detectable Hamming Distance: The accuracy of the associative search depends on the bit precision of ganged-logic design. Using large-size transistors in the ganged logic will result in a faster response to *ML* discharging, thus improving the detection accuracy of the row with the minimum Hamming distance to the input. Figure 8 shows the HD classification accuracy and the energy-delay product (EDP) of SearcHD associative memory, when we change the minimum detectable bits in design from 10 to 90 bits. The results are reported for the activity recognition dataset (UCIHAR). The EDP values are normalized to SearcHD using 10 detectable Hamming distance.

As the graph shows, the design can provide acceptable accuracy when the minimum detectable number of bits is below 32. In this configuration, the associative memory can achieve an EDP improvement of 2.3× when compared to using the design with a 10-bit minimum detectable Hamming distance. That said, ganged logic in low bit precision improves the EDP efficiency while degrading the classification accuracy. For instance, 50-bits and 70-bits minimum detectable Hamming distances can provide 3× and 4.8× EDP improvement as compared to the design with 10-bit detectable Hamming distances, while providing 1% and 3.7% lower than maximum SearcHD accuracy. To find the maximum required precision in CAM circuitry, we cross-checked the distances between all stored class hypervectors. We observed that the distance

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2019.2952544, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

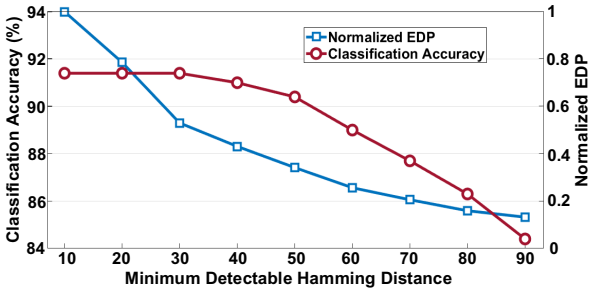


Fig. 8. SearcHD classification accuracy and normalized EDP improvement when the associative memory works in different minimum detectable distances.

between any two stored classes is always higher than 71 bits. In other words, 71 is the minimum Hamming distance which needs to be detected in our design. This feature allows us to relax the bit precision of the analog search sense amplifier which results in further improvement in its efficiency.

Prior work tried to modify the CAM structure in order to enable nearest Hamming distance search capability [21]. However, that design is very sensitive to dimensionality and process variations. Our evaluation on a CAM with 10,000 bits shows that SearcHD associative memory can provide 8 bits Hamming-detectable error under 10% process variations, while work in [21] works with 22 bits Hamming-detectable error. In addition, our proposed approach provides 3.2× higher energy efficiency and 2.0× faster as compared to work in [21]. In Section V, we show the impact of SearcHD associative search on the training/inference efficiency.

F. Accuracy-Efficiency Tradeoff

SearcHD can exploit hypervector dimensions as a parameter to trade efficiency and accuracy. Regardless of the dimension of the model at training, SearcHD can use a model in lower dimensions in order to accelerate SearcHD inference. In HD computing, the dimensions are independent, thus SearcHD can drop any arbitrary dimension in order to accelerate the computation. Figure 9 shows the classification accuracy, normalized energy consumption and normalized execution time of SearcHD when the hypervector dimension changes from D =2000 to 10,000. Our evaluation shows that SearcHD can achieve maximum accuracy using dimensions around $D = 8,000$. In addition, SearcHD with $D = 4000$ ($D = 6000$) can achieve 2.0× and 1.3× (1.3× and 1.2×) higher energy efficiency and speedup than SearcHD with $D = 10,000$ while providing only 2.1% (1.2%) lower classification accuracy.

G. Area/Energy Breakdown

Here, we compare the area and energy breakdown of digital HD computing with SearcHD analog implementation. Figure 10a shows the area occupied by the encoding and associative search modules. In a digital implementation, encoding takes a large amount of chip area, as it requires to encode data points with up to 800 features. In contrast, analog implementation takes significantly lower area in both encoding and associative search modules. The analog majority

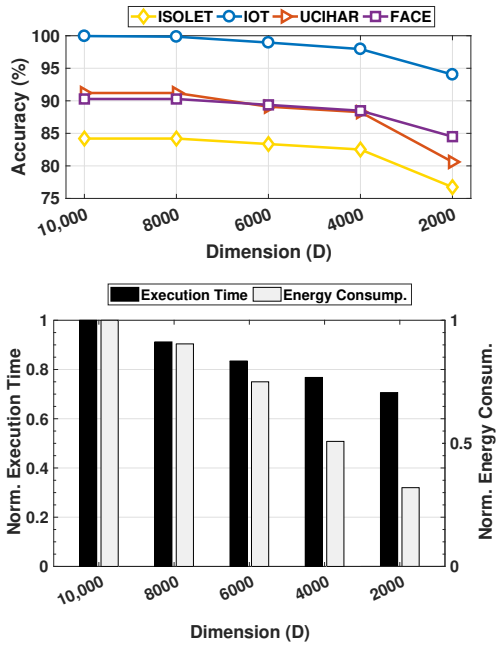


Fig. 9. Impact of dimensionality on SearcHD accuracy and efficiency.

computation in the encoding modules and the analog detector circuit in the associative search module eliminate large circuits for digital accumulation. This results in 6.5× area efficiency of the analog as compared to digital implementation.

Figure 10b shows the area and energy breakdown of the encoding module in digital and analog implementations. In digital, XOR array and accumulator are taking the majority of the area and energy consumption. The accumulator has a higher portion of energy, as this block requires to sequentially add the XOR results. In analog implementation, the majority function dominating the total area and energy, while XOR computation takes about 32% of the area and 16% of energy. This is because the majority module uses a large sense amplifier and exploits switches to split the memory rows (enabling parallel write). Figure 10c shows the area and energy breakdown of the associative search module in both digital and analog implementation. Similar to the encoding module, in digital implementation, XOR array and accumulator are dominating the total area and energy consumption. In analog, the CAM block is dominating the area, as it requires to store all class hypervectors. However, in terms of energy, the detector circuit takes over 64% of total energy. The ADC block takes about 10% area and 7.2% of the energy, as we only require a single ADC block in each associative search module.

H. Hardware Efficiency

To fairly show the advantage of the in-memory implementation, we compare the efficiency of SearcHD with the digital implementation in two configurations: (i) optimized digital implementation with 6.5× larger area than analog, (ii) digital which takes the similar area as analog. The results in Table VII are reported for both training and inference phases, when our implementation provides a similar accuracy

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2019.2952544, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

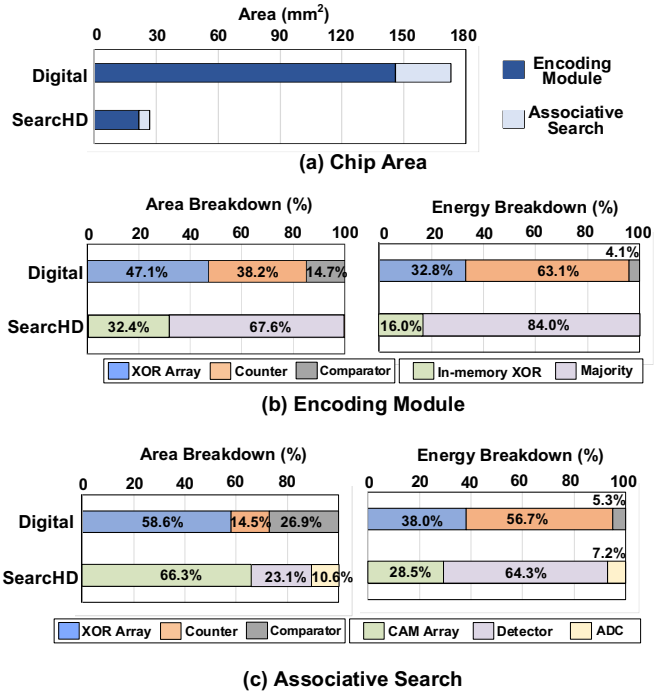


Fig. 10. SearcHD area and energy breakdown. (a) the occupied area by the encoding and associative search modules in digital design and analog SearcHD. (b) area and energy breakdown of the encoding module. (c) area and energy breakdown of the associative search module.

TABLE VII
EFFICIENCY OF SEARCHD AS COMPARED TO DIGITAL IMPLEMENTATION: @ OPTIMIZED AND @ SAME AREA CONFIGURATIONS.

		Speedup				Energy Efficiency			
		ISOLET	*FACE*	*UCIHAR*	*IOT*	*ISOLET*	*FACE*	*UCIHAR*	*IOT*
Training	***Optimized***	20.8	15.8	18.2	47.1	174.5	163.3	206.9	884.9
	@ Area	76.9	66.4	69.7	184.7	218.2	213.9	258.6	1141.5
Inference	***Optimized***	25.6	18.4	18.7	31.3	1093.9	908.0	455.8	2516.1
	@ Area	94.6	70.9	78.5	121.8	1312.7	1125.9	542.4	4554.1

as the baseline digital implementation. Our evaluation shows that analog design provides 25.4× and 357.5× (23.5× and 1243.4×) speedup and energy efficiency as compared to optimized digital implementation during training (inference). In the same area, analog training (inference) efficiency improves to 99.4× and 458.0× (91.4× and 1883.7×) speedup and energy efficiency, respectively.

I. SearcHD & OFF/ON Ratio

In practice, the value of OFF/ON resistance ratio has important impact on the performance of SearcHD functionality. Although we used VTEAM model with 1000 OFF/ON resistance ratio, in practice we may have memristor devices with lower OFF/ON ratio. Using lower OFF/ON ratio has direct impact on the SearcHD performance. In other words, lower ratio makes the functionality of detector circuit more complicated, specially for thresholding functionality. We evaluate the impact of variation on resistance ratio when the OFF/ON ration reduces to 100. Our results shows that this reduction results in 5.8× and 12.5× slower XOR and thresholding functionality, respectively.

VI. CONCLUSION

In this paper, we propose SearcHD, a fully binary HD computing algorithm. SearcHD encodes every data point to HD space with binary elements and performs training by assigning multiple binary hypervectors to each class. During inference, SearcHD searches in the pre-stored class hypervectors for the closest class hypervector to the binary test hypervector in terms of Hamming distance similarity. We accordingly designed an in-memory architecture which can accelerate all SearcHD functionalities in memory. Our experimental evaluation of four practical classification applications shows that SearcHD implemented in-memory can achieve 31.1× and 12.8× (178.7× and 14.1×) energy efficiency and speedup during training (inference) as compared to the state-of-the-art HD computing algorithms while providing 6.5% higher classification accuracy.

ACKNOWLEDGMENT

This work was partially supported by CRISP and ASCENT, two of six centers in JUMP, an SRC program sponsored by DARPA, and also NSF grants #1911095, #1826967, #1730158 and #1527034.

REFERENCES

[1] Y. Xiang, A. Alahi, and S. Savarese, "Learning to track: Online multi-object tracking by decision making," in *2015 IEEE international conference on computer vision (ICCV)*, no. EPFL-CONF-230283, pp. 4705–4713, IEEE, 2015.

[2] D. Amodei, R. Anubhai, E. Battenberg, C. Case, J. Casper, B. Catanzaro, J. Chen, M. Chrzanowski, A. Coates, G. Diamos, *et al.*, "Deep speech 2: End-to-end speech recognition in english and mandarin," in *International Conference on Machine Learning*, pp. 173–182, 2016.

[3] A. Hannun, C. Case, J. Casper, B. Catanzaro, G. Diamos, E. Elsen, R. Prenger, S. Satheesh, S. Sengupta, A. Coates, *et al.*, "Deep speech: Scaling up end-to-end speech recognition," *arXiv preprint arXiv:1412.5567*, 2014.

[4] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[5] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[6] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich, *et al.*, "Going deeper with convolutions," Cvpr, 2015.

[7] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, *et al.*, "Imagenet large scale visual recognition challenge," *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211–252, 2015.

[8] H. Li, T. F. Wu, A. Rahimi, K.-S. Li, M. Rusch, C.-H. Lin, J.-L. Hsu, M. M. Sabry, S. B. Eryilmaz, J. Sohn, *et al.*, "Hyperdimensional computing with 3d vrram in-memory kernels: Device-architecture co-design for energy-efficient, error-resilient language recognition," in *Electron Devices Meeting (IEDM), 2016 IEEE International*, pp. 16–1, IEEE, 2016.

[9] D. Kim, J. Kung, S. Chai, S. Yalamanchili, and S. Mukhopadhyay, "Neurocube: A programmable digital neuromorphic architecture with high-density 3d memory," in *Computer Architecture (ISCA), 2016 ACM/IEEE 43rd Annual International Symposium on*, pp. 380–392, IEEE, 2016.

[10] S. Yu, D. Kuzum, and H.-S. P. Wong, "Design considerations of synaptic device for neuromorphic computing," in *Circuits and Systems (ISCAS), 2014 IEEE International Symposium on*, pp. 1062–1065, IEEE, 2014.

[11] M. Imani, Y. Kim, S. Riazi, J. Messerly, P. Liu, F. Koushanfar, and T. Rosing, "A framework for collaborative learning in secure high-dimensional space," in *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*, pp. 435–446, IEEE, 2019.

[12] P. Kanerva, "Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors," *Cognitive Computation*, vol. 1, no. 2, pp. 139–159, 2009.

[13] M. Hersche, T. Rellstab, P. D. Schiavone, L. Cavigelli, L. Benini, and A. Rahimi, "Fast and accurate multiclass inference for mi-bcis using large multiscale temporal and spectral features," *arXiv preprint arXiv:1806.06823*, 2018.

[14] O. Rasanen and J. Saarinen, "Sequence prediction with sparse distributed hyperdimensional coding applied to the analysis of mobile phone use patterns," *IEEE Transactions on Neural Networks and Learning Systems*, vol. PP, no. 99, pp. 1–12, 2015.

[15] M. Imani, C. Huang, D. Kong, and T. Rosing, "Hierarchical hyperdimensional computing for energy efficient classification," in *Proceedings of the 55th Annual Design Automation Conference*, p. 108, ACM, 2018.

[16] A. Rahimi, A. Tchouprina, P. Kanerva, J. d. R. Millán, and J. M. Rabaey, "Hyperdimensional computing for blind and one-shot classification of eeg error-related potentials," *Mobile Networks and Applications*, pp. 1–12, 2017.

[17] F. Montagna, A. Rahimi, S. Benatti, D. Rossi, and L. Benini, "Pulp-hd: accelerating brain-inspired high-dimensional computing on a parallel ultra-low power platform," *arXiv preprint arXiv:1804.09123*, 2018.

[18] 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 *2019 56th ACM/IEEE Design Automation Conference (DAC)*, pp. 1–6, IEEE, 2019.

[19] M. Imani, J. Messerly, F. Wu, W. Pi, and T. Rosing, "A binary learning framework for hyperdimensional computing," in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 126–131, IEEE, 2019.

[20] A. Rahimi, P. Kanerva, and J. M. Rabaey, "A robust and energy-efficient classifier using brain-inspired hyperdimensional computing," in *Proceedings of the 2016 International Symposium on Low Power Electronics and Design*, pp. 64–69, ACM, 2016.

[21] M. Imani, A. Rahimi, D. Kong, T. Rosing, and J. M. Rabaey, "Exploring hyperdimensional associative memory," in *High Performance Computer Architecture (HPCA), 2017 IEEE International Symposium on*, pp. 445–456, IEEE, 2017.

[22] M. Imani, S. Gupta, Y. Kim, and T. Rosing, "Floatpim: In-memory acceleration of deep neural network training with high precision," in *Proceedings of the 46th International Symposium on Computer Architecture*, pp. 802–815, ACM, 2019.

[23] P. Chi, S. Li, C. Xu, T. Zhang, J. Zhao, Y. Liu, Y. Wang, and Y. Xie, "Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory," in *Proceedings of the 43rd International Symposium on Computer Architecture*, pp. 27–39, IEEE Press, 2016.

[24] A. Shafiee, A. Nag, N. Muralimanohar, R. Balasubramonian, J. P. Strachan, M. Hu, R. S. Williams, and V. Srikumar, "Isaac: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 14–26, 2016.

[25] J. Ahn, S. Yoo, O. Mutlu, and K. Choi, "Pim-enabled instructions: A low-overhead, locality-aware processing-in-memory architecture," in *Computer Architecture (ISCA), 2015 ACM/IEEE 42nd Annual International Symposium on*, pp. 336–348, IEEE, 2015.

[26] J. Ahn, S. Hong, S. Yoo, O. Mutlu, and K. Choi, "A scalable processing-in-memory accelerator for parallel graph processing," in *Computer Architecture (ISCA), 2015 ACM/IEEE 42nd Annual International Symposium on*, pp. 105–117, IEEE, 2015.

[27] S. Li, C. Xu, Q. Zou, J. Zhao, Y. Lu, and Y. Xie, "Pinatubo: A processing-in-memory architecture for bulk bitwise operations in emerging non-volatile memories," in *Design Automation Conference (DAC), 2016 53nd ACM/EDAC/IEEE*, pp. 1–6, IEEE, 2016.

[28] M. Imani, M. Samragh, Y. Kim, S. Gupta, F. Koushanfar, and T. Rosing, "Rapidnn: In-memory deep neural network acceleration framework," *arXiv preprint arXiv:1806.05794*, 2018.

[29] A. Boroumand, S. Ghose, B. Lucia, K. Hsieh, K. Malladi, H. Zheng, and O. Mutlu, "Lazypim: An efficient cache coherence mechanism for processing-in-memory," *IEEE Computer Architecture Letters*, 2017.

[30] X. Yin, X. Chen, M. Niemier, and X. S. Hu, "Ferroelectric fets-based nonvolatile logic-in-memory circuits," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 1, pp. 159–172, 2019.

[31] M. Imani, S. Gupta, and T. Rosing, "Ultra-efficient processing in-memory for data intensive applications," in *Proceedings of the 54th Annual Design Automation Conference 2017*, p. 6, ACM, 2017.

[32] T. F. Wu, H. Li, P.-C. Huang, A. Rahimi, J. M. Rabaey, H.-S. P. Wong, M. M. Shulaker, and S. Mitra, "Brain-inspired computing exploiting carbon nanotube fets and resistive ram: Hyperdimensional computing case study," in *Solid-State Circuits Conference-(ISSCC), 2018 IEEE International*, pp. 492–494, IEEE, 2018.

[33] M. Imani, A. Rahimi, and T. S. Rosing, "Resistive configurable associative memory for approximate computing," in *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1327–1332, IEEE, 2016.

[34] p. kanerva, "What we mean when we say" what's the dollar of mexico?": Prototypes and mapping in concept space," in *AAAI fall symposium: quantum informatics for cognitive, social, and semantic processes*, vol. 1036, Citeseer, 2010.

[35] P. Kanerva, J. Kristofersson, and A. Holst, "Random indexing of text samples for latent semantic analysis," in *Proceedings of the 22nd annual conference of the cognitive science society*, vol. 1036, Citeseer, 2000.

[36] A. Joshi, J. Halseth, and P. Kanerva, "Language geometry using random indexing," *Quantum Interaction 2016 Conference Proceedings*, In press.

[37] A. Rahimi, P. Kanerva, and J. M. Rabaey, "A robust and energy efficient classifier using brain-inspired hyperdimensional computing," in *Low Power Electronics and Design (ISLPED), 2016 IEEE/ACM International Symposium on*, August 2016.

[38] F. R. Najafabadi, A. Rahimi, P. Kanerva, and J. M. Rabaey, "Hyperdimensional computing for text classification," *Design, Automation Test in Europe Conference Exhibition (DATE), University Booth*, 2016.

[39] A. Rahimi, S. Benatti, P. Kanerva, L. Benini, and J. M. Rabaey, "Hyperdimensional biosignal processing: A case study for emg-based hand gesture recognition," in *Rebooting Computing (ICRC), IEEE International Conference on*, pp. 1–8, IEEE, 2016.

[40] O. Räsänen and S. Kakouros, "Modeling dependencies in multiple parallel data streams with hyperdimensional computing," *IEEE Signal Processing Letters*, vol. 21, no. 7, pp. 899–903, 2014.

[41] S. Jockel, "Crossmodal learning and prediction of autobiographical episodic experiences using a sparse distributed memory," 2010.

[42] M. Imani, D. Kong, A. Rahimi, and T. Rosing, "Voicehd: Hyperdimensional computing for efficient speech recognition," in *International Conference on Rebooting Computing (ICRC)*, pp. 1–6, IEEE, 2017.

[43] "Uci machine learning repository." http://archive.ics.uci.edu/ml/datasets/ISOLET.

[44] D. Ayres-de Campos, J. Bernardes, A. Garrido, J. Marques-de Sa, and L. Pereira-Leite, "Sisporto 2.0: a program for automated analysis of cardiotocograms," *Journal of Maternal-Fetal Medicine*, vol. 9, no. 5, pp. 311–318, 2000.

[45] 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 *International workshop on ambient assisted living*, pp. 216–223, Springer, 2012.

[46] Y. Mirsky, T. Doitshman, Y. Elovici, and A. Shabtai, "Kitsune: an ensemble of autoencoders for online network intrusion detection," *arXiv preprint arXiv:1802.09089*, 2018.

[47] X. Yin, K. Ni, D. Reis, S. Datta, M. Niemier, and X. S. Hu, "An ultra-dense 2fefet tcam design based on a multi-domain fefet model," *IEEE Transactions on Circuits and Systems II: Express Briefs*, 2018.

[48] P. Kanerva, J. Kristofersson, and A. Holst, "Random indexing of text samples for latent semantic analysis," in *Proceedings of the 22nd annual conference of the cognitive science society*, vol. 1036, Citeseer, 2000.

[49] S. Kvatinsky, D. Belousov, S. Liman, G. Satat, N. Wald, E. G. Friedman, A. Kolodny, and U. C. Weiser, "Magicmemristor-aided logic," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 61, no. 11, pp. 895–899, 2014.

[50] K. Schultz, R. J. Francis, and K. C. Smith, "Ganged cmos: Trading standby power for speed," *IEEE Journal of Solid-State Circuits*, vol. 25, no. 3, pp. 870–873, 1990.

[51] Z. Wei, Y. Katoh, S. Ogasahara, Y. Yoshimoto, K. Kawai, Y. Ikeda, K. Eriguchi, K. Ohmori, and S. Yoneda, "True random number generator using current difference based on a fractional stochastic model in 40-nm embedded reram," in *Electron Devices Meeting (IEDM), 2016 IEEE International*, pp. 4–8, IEEE, 2016.

[52] Y. Wang, W. Wen, H. Li, and M. Hu, "A novel true random number generator design leveraging emerging memristor technology," in *Proceedings of the 25th edition on Great Lakes Symposium on VLSI*, pp. 271–276, ACM, 2015.

[53] S. H. Jo, K.-H. Kim, and W. Lu, "Programmable resistance switching in nanoscale two-terminal devices," *Nano letters*, vol. 9, no. 1, pp. 496–500, 2008.

[54] H. Jiang, D. Belkin, S. E. Savelev, S. Lin, Z. Wang, Y. Li, S. Joshi, R. Midya, C. Li, M. Rao, *et al.*, "A novel true random number generator based on a stochastic diffusive memristor," *Nature communications*, vol. 8, no. 1, p. 882, 2017.

[55] "Tsmc 45 nm technology." https://www.synopsys.com/dw/emllselector.php?f=TSMC&n=45&s=wMoRVw.

[56] S. Kvatinsky, M. Ramadan, E. G. Friedman, and A. Kolodny, "Vteam: A general model for voltage-controlled memristors," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 62, no. 8, pp. 786–790, 2015.

[57] Y. Kim, M. Imani, and T. Rosing, "Orchard: Visual object recognition accelerator based on approximate in-memory processing," in *Computer-Aided Design (ICCAD), 2017 IEEE/ACM International Conference on*, pp. 25–32, IEEE, 2017.