SparseHD: Algorithm-Hardware Co-Optimization for Efficient High-Dimensional Computing

Mohsen Imani, Sahand Salamat, Behnam Khaleghi, Mohammad Samragh, Farinaz Koushanfar, Tajana Rosing
University of California San Diego, La Jolla, CA 92093, USA
{moimani, sasalama, bkhalegh, msamragh, fkoushanfar, tajana}@ucsd.edu

***Abstract*—Hyperdimensional (HD) computing is gaining traction as an alternative light-way machine learning approach for cognition tasks. Inspired by the neural activity patterns of the brain, HD computing performs cognition tasks by exploiting long-size vectors, namely *hypervectors*, rather than working with scalar numbers as used in conventional computing. Since a hypervector is represented by thousands of dimensions (elements), the majority of prior work assume binary elements to simplify the computation and alleviate the processing cost. In this paper, we first demonstrate that the dimensions need to have more than one bit to provide an acceptable accuracy to make HD computing applicable to real-world cognitive tasks. Increasing the bit-width, however, sacrifices energy efficiency and performance, even when using low-bit integers as the hypervector elements.**

To address this issue, we propose a framework for HD acceleration, dubbed SparseHD, that leverages the advantages of sparsity to improve the efficiency of HD computing. Essentially, SparseHD takes account of statistical properties of a trained HD model and drops the least effective elements of the model, augmented by iterative retraining to compensate the possible quality loss raised by sparsity. Thanks to the bit-level manipulability and abounding parallelism granted by FPGAs, we also propose a novel FPGA-based accelerator to effectively utilize the advantage of sparsity in HD computation. We evaluate the efficiency of our framework for practical classification problems. We observe that SparseHD makes the HD model up to 90% sparse while affording a minimal quality loss (less than 1%) compared to the non-sparse baseline model. Our evaluation shows that, on average, SparseHD provides 48.5× and 15.0× lower energy consumption and faster execution as compared to the AMD R390 GPU implementation.

I. Introduction

Machine learning algorithms have become ubiquitous as they have demonstrated effectiveness in various tasks, e.g., object tracking, speech recognition, and image classification [1]–[5]. This has been further accentuated with the emergence of the Internet of Things (IoT), where different applications run learning algorithms to perform cognitive tasks. However, the massive streams of data produced by the sensory and embedded devices pose serious processing challenges due to limited resources [6]–[8], which makes it inevitable to devise alternative computing methods that can efficiently process a large amount of data with an affordable cost.

Brain-inspired Hyperdimensional (HD) computing has been proposed as an alternative computing method that performs the cognitive tasks using a more light-weight approach [9]. HD computing is established on the fact that an organic brain processes *patterns of neural activity* which are not readily associated with numerical numbers [9], [10]. Recent research, instead, have exploited high dimensional vectors (e.g., more than a thousand dimension), called *hypervectors*, to represent the neural activities, and shown successful progress for many cognitive tasks [11]–[15]. Specifically, compared to conventional learning algorithms, (a) HD offers an efficient learning strategy without over-complex computation steps such as back propagation in neural networks. Note that even binary neural networks are very costly to train and their advantage over non-binary models is limited to test time [16]–[19]. (b) HD computing builds upon a well-defined set of operations with random HD vectors which makes the learning process extremely robust in the presence of hardware failures and noise. In fact, alternatives counterparts such as neural networks are shown to be vulnerable to adversatial noise patterns [20], [21]. (c) HD can be easily applied to diverse problems and applications including language recognition [22], [23], voice recognition [14], DNA sequencing [24], activity recognition [25], clustering [26], and collaborative learning.

In HD computing, training data are encoded (i.e., mapped to a high-dimensional space) and aggregated to form a set of hypervectors, called an *HD model*, by light-weight computation steps. In case of classification, each hypervector represents a separate class. The similarity of a given input (also encoded to a hypervector) with the class hypervectors determines the model prediction. Most of the previous HD works exploit binarized hypervectors to reduce the computations and memory intensity of HD computing [13], [14], [27]–[29]. As we will show later, using non-binary hypervectors with real-valued elements improves the accuracy by more than 50% for a specific task. However, it requires a higher computation cost to perform a large number of multiply-add operations, compared to the binarized HD that mainly uses bitwise operations.

In this paper, we present a robust and efficient solution that throttles the computational load while preserving the numeric precision of the non-binarized hypervectors, further raising the profile of HD computing. The proposed HD acceleration framework, called SparseHD, explores the prospect of sparsity in the hypervectors to improve the HD computing efficiency. SparseHD takes advantage of statistical properties of HD models to make the trained hypervectors sparse without losing the quality of inference (prediction). It reformulates the training phase of HD model to enforce sparsity by eliminating the least impactful features in the trained hypervectors. We examine two approaches for enforcing sparsity: (a) class-wise sparsity which independently sparsifies hypervectors of each class by discarding the elements that have minimal impact on the results, and (b) dimension-wise sparsity that identifies and discards the inconsequential (non-informative) dimensions

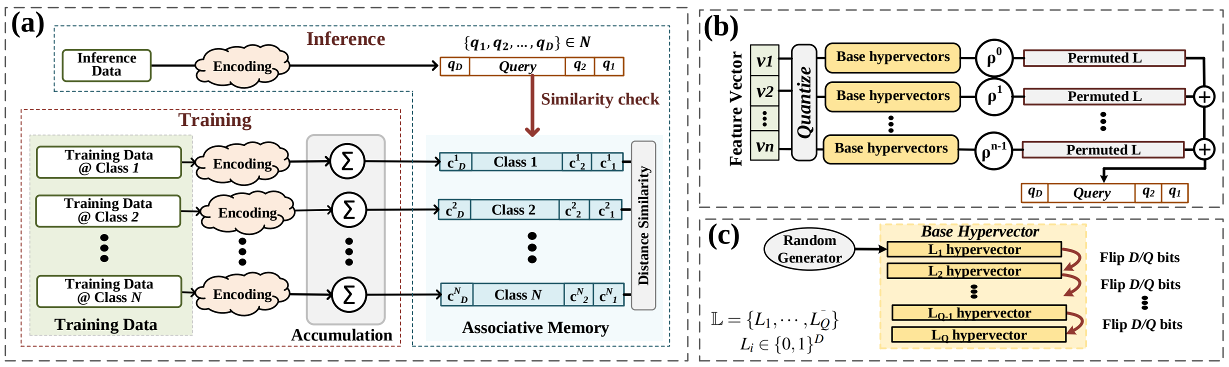


Fig. 1. (a) Overview of the HD classification consisting of encoding and associative memory modules. (b) The encoding module maps a feature vector to a high-dimensional space using pre-generated base hypervectors. (c) Generating the base hypervectors.

shared across all learned hypervectors. Thereafter, SparseHD realizes an efficient FPGA implementation of the proposed sparse HD model. Considering the nature of operations in HD computing, FPGA is the most appealing solution for HD acceleration due to the great degree of parallelism controllable in a *fine-grained* manner, afforded intrinsically by these devices [30]. **The main contributions of this paper are as follows**.

- We develop the first sparse HD computing method that enables sparsity on the trained HD model. We also propose an automated technique which iteratively retrains HD models to compensate the potential quality loss that might be incurred by model sparsity.
- We implement a user-friendly platform for FPGA implementation of sparse HD computation that supports both the dimension-wise and class-wise sparse models. Our FPGA accelerator is hand-crafted in a pipelined structure to effectively utilize the FPGA resources to maximize performance.
- We perform extensive evaluations on practical classification problems. Compared to AMD R390 baseline implementation, SparseHD implemented on Kintex-7 FPGA KC705 Evaluation Kit achieves on average 45.5× lower energy consumption and 15.0× faster execution time. In addition, ensuring the quality loss of less than 0.5% and 1.5%, SparseHD achieves 11.4× and 49.7× Energy-Delay Product (EDP) improvement as compared to the FPGA implementation of baseline HD.

II. Preliminary

HD provides a general model of computing that can be applied to different types of learning problems. Fig. 1(a) shows the overview of HD computing architecture for a demonstrative classification problem. The HD architecture comprises an encoding module and an associative memory, a.k.a. similarity check. The encoding module maps each input data to a hypervector. During the training, all the training inputs corresponding to a particular class are encoded and combined together to generate a *class hypervector*. There is a class hypervector for each class which is stored in an associative memory. In the inference (prediction) phase, an unlabeled input data is mapped to a *query hypervector* using the same encoding module used for training. The query hypervector is then compared with all class hypervectors to determine the classification result.

A. Encoding Module

The encoding module works on the pre-processed data, i.e., extracted features, which vary from application to application. For instance, a voice signal might be first transferred to Mel-Frequency Cepstral Coefficients (MFCCs) feature vector [31]. Fig. 1(b) illustrates how the encoding module maps a single input data to high-dimensional space of $\mathcal{D}$ (e.g., 10,000) elements. Consider a single input data represented by feature vector $\vec{V}_{fv} = \langle v_1, v_2, \cdots, v_n \rangle$, wherein $n \ll \mathcal{D}$ is the number of features per input, so is application-dependent. The encoding essentially comprises two main steps as follows.

(1) Generating base hypervectors: Each feature value $v_i \in [v_{min}, v_{max}]$ in the input feature vector $\vec{V}_{fv}$ can have different discrete or continuous values, that require to be quantized to Q levels, denoted by $\mathcal{L} = \{l_1, l_2, \cdots, l_Q\}$ with l_1 and l_Q corresponding to v_{min} and v_{max}, respectively. Each scalar l_i corresponds to a $\mathcal{D}$-dimensional binary hypervector, called *base hypervector*. The base hypervectors have to maintain the *proximity* of the levels, i.e., if the values of l_i and l_j (hence, v_i and v_j) are relatively close, the corresponding base hypervectors $\vec{L}_i$ and $\vec{L}_j$ need to have relatively small Hamming distance. Consequently, $\vec{L}_1$ and $\vec{L}_Q$ should be orthogonal. Therefore, as shown in Fig. 1(c), to create the entire set of base hypervectors, the first seed hypervector $\vec{L}_1$ associated with l_1 is created by random binary elements. Each of the subsequent level base hypervector is then created by flipping specific $^D/_Q$ of dimensions. This leads $\vec{L}_Q$ to be orthogonal with respect to $\vec{L}_1$ while similar feature values have similar base hypervectors. For data with quantized bases, e.g., text and DNA sequences, the level hypervectors do not need to have proximity correlation [32].

(2) Element-wise hypervector mapping: Once all the base hypervectors are generated, each of the n elements of the input feature vector $\vec{V}_{fv}$ are independently quantized and mapped to the corresponding base hypervector according to its level. In the last step, the n base hypervector of input $\vec{V}_{fv}$ need to be combined into a single representative hypervector. The naïve approach would be to aggregate (add up) all base hypervectors of the elements but such an approach does not take account of the spatial and/or temporal distance (i.e., index of each feature) of the features. To differentiate the impact of feature indexes, we employ *permutation*. From the distribution of random binary values we know that permuting different indexes keeps the vectors nearly orthogonal [33], i.e., $\delta(\vec{L}, \mathcal{P}_{\vec{L}}^{(i)}) \simeq {}^{D}\!/_{2}$, with δ standing for Hamming distance and $\mathcal{P}_{\vec{L}}^{(i)}$ denoting i-bits rotational permutation of vector $\vec{L}$. The orthogonality of a base hypervectors and its permuted pair is assured as long as the hypervector dimensionality is long enough compared to the number of features ($\mathcal{D} \gg n$). Hence, the aggregation of n hypervectors corresponding to each feature index is obtained via the following equation, illustrated also by Fig. 1(b).

$$\vec{\mathcal{H}} = \vec{L}_{v_1} + \mathcal{P}_{\vec{L}_{v_2}}^{(1)} + \cdots + \mathcal{P}_{\vec{L}_{v_n}}^{(n-1)} = \sum_{i=1}^{n} \mathcal{P}_{\vec{L}_{v_i}}^{(i-1)} \qquad (1)$$

$\vec{\mathcal{H}}$ is the non-binary encoded hypervector of input $\vec{V}_{fv}$ and $\vec{L}_{v_i}$ is the binary base hypervector of (the level of) feature v_i.

B. Model Training

Training of an HD model consists of generating the base hypervectors, which is done just once, and encoding every input feature vector, as explained above. Thereafter, the encoded hypervectors belonging to the same prediction class (label) are accumulated to build up the class's hypervector. Thus, for the class hypervector $\vec{\mathcal{C}}$ with label i, we have:

$$\vec{\mathcal{C}^i} = \langle c_{\mathcal{D}}, \cdots c_1 \rangle = \sum_j \vec{\mathcal{H}}_j^i \qquad , \vec{\mathcal{H}^i} = \langle h_{\mathcal{D}}, \cdots h_1 \rangle \qquad (2)$$

$\vec{\mathcal{H}^i}$ indicates the encoded hypervector of an input with class (prediction label) i. As an instance, in a face detection task, the trainer adds all hypervectors which have the 'face' tag and 'non-face' tags in two different class hypervectors.

Binarized model: The additions involved in HD training are element-wise and result in class hypervectors with non-binary dimension elements, i.e., $\vec{\mathcal{C}} \in \mathbb{N}^{\mathcal{D}}$. To perform the classification using binary hypervectors, a threshold function needs to be applied on the non-binary class hypervectors:

$$\mathcal{T}(\vec{\mathcal{C}}, \tau) = \langle c'_{\mathcal{D}}, \cdots c'_1 \rangle \text{ where } c'_i = \begin{cases} 0, & \text{if } c_i < \tau \\ 1, & \text{otherwise.} \end{cases} \qquad (3)$$

That is, for every element c_i of the class hypervector, it is checked whether the same element in at least τ out of its k building hypervectors $\vec{\mathcal{H}}$ was 1, so usually $\tau = \frac{k}{2}$.

C. Inference (Test/Prediction)

During the inference, an input data is encoded to a so-called *query hypervector* using the same encoding scheme used for training as explained above. The associative memory (a.k.a similarity check) is responsible to compare the query hypervector with all class hypervectors to find out the one with the highest similarity (see Fig. 1(a)). In the context of binarized HD model, Hamming distance is an inexpensive and suitable metric of similarity, while non-binary class hypervectors need to use *cosine* similarity. The cosine similarity can be expressed as $cos(\vec{\mathcal{H}}, \vec{\mathcal{C}^i}) = \frac{\vec{\mathcal{H}} \cdot \vec{\mathcal{C}^i}}{\|\vec{\mathcal{H}}\| \cdot \|\vec{\mathcal{C}^i}\|}$, where $\vec{\mathcal{H}} \cdot \vec{\mathcal{C}^i}$ indicates dot product between the hypervectors, and $\|\vec{\mathcal{H}}\|$ and $\|\vec{\mathcal{C}^i}\|$ show the magnitudes of the query and i^{th} class hypervector. As query $\vec{\mathcal{H}}$ is common between all the candidate classes, we can ignore $\|\vec{\mathcal{H}}\|$ when finding the maximum *relative* similarity. The magnitude of each class hypervector, $\|\vec{\mathcal{C}}\| = \sum c_i^2$ can be computed once offline after the training, which simplifies the cosine similarity to a dot product between two hypervectors at inference, that can be computed in much lower cost: $similarity(\vec{\mathcal{C}}, \vec{\mathcal{H}}) = \sum_{i=1}^{\mathcal{D}} c_i \cdot h_i$

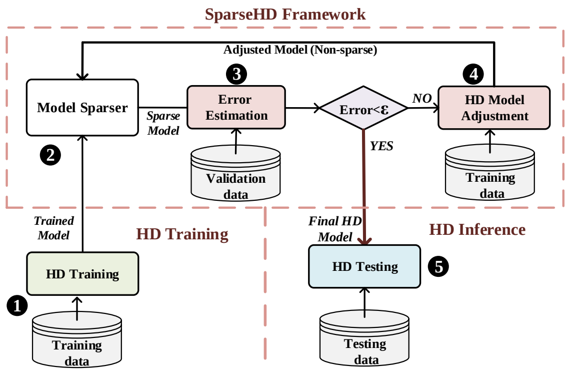


Fig. 2. Overview of SparseHD algorithmic framework enabling sparsity in HD computing model.

D. Binarization Accuracy Loss

Most existing HD computing methods use binary class hypervectors to eliminate costly cosine operation [13], [14], [27], [34]. Our result shows that, HD computing accuracy using a binary model is significantly lower than a non-binarized model. For example, for the face detection task, binarized HD achieves a classification accuracy of 38.9%, which is far lower than 96.1% of the non-binarized counterpart. However, the non-binary HD rises from the costly cosine similarity metric that involves a large number of additions/multiplications, making it less desirable as a light-weight classifier. Our evaluation on four practical applications listed in Section V shows that the non-binary HD model delivers 17.5% better prediction accuracy though it is 6.5× slower in computation.

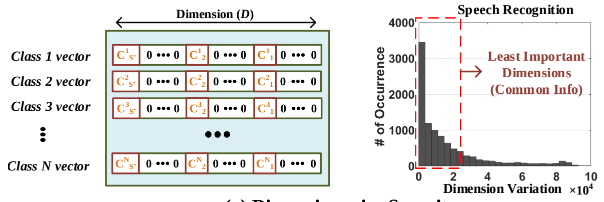
III. Model Sparsification

A. Overview

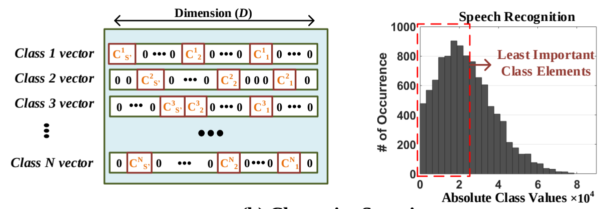
Fig. 2 shows the overview of the proposed SparseHD framework. SparseHD takes a trained HD model in non-binary dense representation as an input (❶). For each class hypervector, the *model sparser* drops S% of each class elements (❷). The classification accuracy of the sparse model is examined on the validation dataset, which is a part of the original training dataset. Thereafter, SparseHD compares the accuracy of HD with the sparse and dense model to calculate the quality loss due to model sparsity (❸). For errors larger than a pre-defined threshold ε, SparseHD adjusts the HD model by retraining the HD based on the sparsity constraint (❹). The model adjustment may change the sparsity of class hypervectors, thus the *Model Sparser* resets the sparsity of the HD model to the desired level. The model adjustment and sparsification process repeats iteratively until the convergence condition is satisfied.

B. Model Sparsifier

We propose two techniques to sparsify the HD computing model: dimension-wise and class-wise sparsity. The dimension-wise technique sparsifies the trained HD models by dropping the same dimensions for all existing classes, while the class-wise method makes each class hypervector sparse individually. Fig. 3 shows an example of class elements using



(a) Dimension-wise Sparsity



(b) Class-wise Sparsity

Fig. 3. (a) An example of the SparseHD dimension-wise sparsity model and distribution of the values variation ($\Delta\mathcal{V}$) in all dimensions of the class hypervectors. (b) An example of the trained SparseHD class-wise sparsity model and the distribution of the absolute class values in a trained model.

class-wise and dimension-wise sparsity. In the following, we explain what's the motivation behind each of these methods and how the sparsity can be applied to a trained HD model.

(1) Dimension-wise sparsity: The goal of HD computing at inference is to find a class hypervector with the highest cosine similarity to the query hypervector, which is *relative* among the class hypervectors. We observe that not all dimensions of the class hypervectors have useful information that can distinguish one class from others. In several dimensions, all class hypervectors store common information shared among all classes, which add relatively similar weight to all classes in calculating the cosine similarity. To enable dimension-wise sparsity in HD computing, our framework measures the changes in the class elements in each dimension. The following equation shows the variation in the j^{th} dimension of the class hypervectors:

$$\Delta\mathcal{V}_j = \mathbf{max}\{c_j^1, \ldots, c_j^N\} - \mathbf{min}\{c_j^1, \ldots, c_j^N\} \qquad j \in \{1, 2, \ldots, \mathcal{D}\} \tag{4}$$

where c_j^i denotes j^{th} element of the i^{th} class hypervector.

After obtaining the variation of dimensions ($\Delta\mathcal{V}_j$s), SparseHD selects the dimensions with the lowest $\Delta\mathcal{V}$ as the best candidates to be dropped from the HD model as they have the least impact on differentiating the classes. Fig. 3(a) shows the histogram distribution of the $\Delta\mathcal{V}$ in all dimensions of the class hypervectors for speech recognition (ISOLET) dataset with 26 classes. Many dimensions have low variation in values across the classes, i.e., they have similar values in those dimensions. We obtained a similar $\Delta\mathcal{V}$ distribution for the six applications (reported in Section V), mainly because the feature vectors have many similar patterns in the original domain, which get distributed uniformly in high-dimensional space. For S% sparsity, we select $\mathtt{S} \times \mathcal{D}$ dimensions with the least $\Delta\mathcal{V}$ and discards those class entries of these dimensions.

(2) Class-wise sparsity: In class-wise sparsity, the goal is to drop the elements of each individual class that have the least impact on the cosine similarity. While calculating the cosine similarity, the elements of a query hypervector are input dependent and can change from one input to another one. Due to the randomness of HD base hypervectors, averaging the query hypervectors results in a hypervector with a uniform distribution of values in all dimensions. Using this assumption, class-wise sparsity needs to find the best class elements that can be dropped while having minimal impact on the cosine similarity. Fig. 3(b) shows the distribution of the absolute class values in a single class hypervector for speech recognition after training. The graph visualizes the best candidates which can be dropped from a single class hypervector. In fact, the values with the least absolute values are the best candidates which can be dropped while causing least impact on the cosine similarity. For example, for the i^{th} class hypervector, we select S% elements with minimum absolute value as follows.

$$\mathbf{min}\{c_{\mathcal{D}}^i, \;\ldots,\; c_2^i,\; c_1^i\}_{\mathcal{S}} \qquad , i \in \{1, 2, \ldots, |\vec{\mathcal{C}}|\} \tag{5}$$

To make a model with S% sparsity, SparseHD makes $\mathtt{S} \times \mathcal{D}$ elements of each class hypervector zero. This method reduces the number of required multiplication and addition operations by ensuring each class hypervector will not have more than $(1 - \mathtt{S}) \times \mathcal{D}$ non-zero elements. Provided appropriate hardware support, the sparsity of class hypervectors can significantly accelerate the performance of HD.

C. Model Adjustment

Sparsifying may affect the HD classification accuracy since the design was not originally trained to work with sparse hypervectors. Our design estimates the error rate of the sparse model by checking its average accuracy over the validation data and compares it with the baseline HD model, $\Delta e = e_{Baseline} - e_{Sparse}$. To compensate for the quality loss due to model sparsity, we adjust the model based on the new constraints. Model adjustment is similar to training procedure and its goal is to enhance the sparse model to provide higher accuracy over training data. HD looks for the similarity of each input hypervector to all stored class hypervectors; (i) If a query hypervector, $\vec{\mathcal{H}}$, is correctly classified by the current model, our design does not change the model. (ii) However, if it is wrongly matched with the i^{th} class hypervector ($\vec{\mathcal{C}^i}$) while it actually belongs to class $\vec{\mathcal{C}^j}$, our retraining procedure subtracts the query hypervector from the i^{th} class hypervector and adds it to class $\vec{\mathcal{C}^j}$ hypervector:

$$\vec{\mathcal{C}^i_{new}} = \vec{\mathcal{C}^i} - \vec{\mathcal{H}} \qquad \text{and} \qquad \vec{\mathcal{C}^j_{new}} = \vec{\mathcal{C}^j} + \vec{\mathcal{H}} \tag{6}$$

After adjusting the model over training data, the class elements may not retain their S% sparsity. Therefore, the framework repeats the algorithm by dropping the inconsequential elements of the new class to ensure the S% sparsity in class hypervectors and estimates the classification error rate over validation data again, until an error rate, ε, is satisfied or a predefined number of iterations passed.

IV. FPGA Implementation

The baseline HD computing algorithm involves a huge amount of multiplications that can be effectively parallelized on GPU or FPGA platforms. However, GPUs are optimized for dense computations with regular data access patterns and cannot benefit much from a sparse model. On the other hand, due to the resource constraints of FPGA, the encoding module

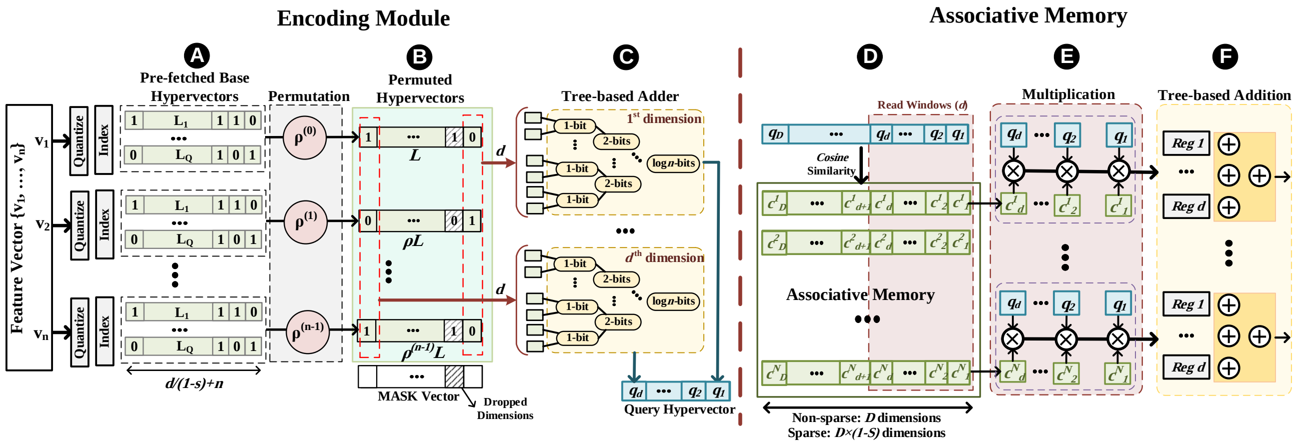


Fig. 4. FPGA implementation of the encoding module and associative memory for SparseHD with dimension-wise sparsity.

and associative memory cannot simultaneously process all dimensions of hypervectors. As a result, we need to segregate and process the dimensions in batches of d. This, however, imposes a significant latency overhead. Thus, we implement a pipeline architecture which hides the delay of the encoding module. In our implementation, at the time encoding module generates d dimensions of the query hypervector, the associative memory module performs the similarity check on another d dimensions that were encoded in the previous cycle.

A. Encoding Implementation

To accelerate the encoding process, the FPGA stores all the base hypervectors ($L \in \{0,1\}^{\mathcal{D}}$) in Block RAMs. In encoding, the maximum number of required permutations is $n-1$ (for the last feature v_n), where n is the number of features. Thus, to calculate the first dimension (h_1) of the query hypervector, $\vec{\mathcal{H}}$, we only need to access 1^{st} to n^{th} dimensions/bits of the base hypervectors (see Fig. 4B). Accordingly, to generate the first d dimensions of the query hypervector, the encoding module requires 1 to "$\mathtt{d}+n$" indexes of the base hypervectors as the $\mathtt{d}^{th}$ dimension requires d to "$\mathtt{d}+n$" indexes. Similarly, for the i^{th} cycle, our implementation only requires to prefetch the indexes "$i \times \mathtt{d}$" to "$i \times \mathtt{d}+n$" of the base hypervectors.

Since the base hypervectors are in binary, the dimension-wise addition of the permuted hypervectors is similar to the popcount operation. SparseHD implements a tree-based pipeline structure to add up all the bits in the same dimension. This structure uses a 1-bit adder in the first stage and then increases the bit-width of the adders by one bit at each stage. In the last stage ($\log n^{th}$ stage), a single $\log n$-bit adder calculates the final result of addition of all n hypervectors (Fig. 4C). To parallelize the addition on all dimensions, SparseHD implements multiple instances of the same tree-based adder, i.e., one adder-tree per every fetched dimension. Note that to balance the pipeline between the encoding module and the associative memory, the number of query elements generated by the encoding module should not exceed the number of elements processed by the associative memory at each cycle.

The main implementation challenge in encoding the dimension-wise sparse HD is to skip generating the dimensions of the query hypervector that correspond to an inconsequential dimension in the class hypervectors (that have been

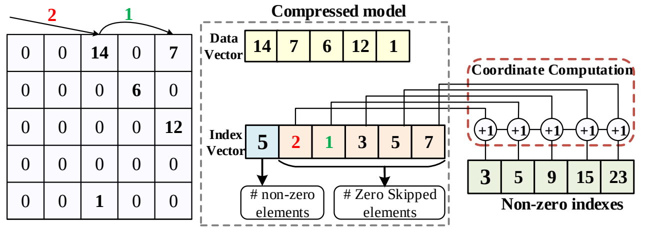


Fig. 5. Compressed format of an 80% matrix and coordinate computation required to decompress the HD model.

dropped out). The sparsity pattern of the d dimension batches fetched at each cycle is not similar, so simply avoiding the addition of predetermined dimensions of the batches is not effective. For this purpose, at each cycle, $\frac{\mathtt{d}}{1-\mathtt{S}}+n$ dimensions of the base hypervectors are fetched and permuted to generate $\frac{\mathtt{d}}{1-\mathtt{S}}$ dimensions of the (permuted) base hypervectors. We use a mask module, indicated by *MASK Vector* in Fig. 4B, that stores the indexes of the effective query elements (for which the corresponding class dimensions are actually used) to generate the d effective dimensions of the query hypervector. This module basically stores the d effective elements of the prefetched hypervectors in an intermediate memory, and passes them to the d adder blocks in the next cycle. Otherwise, it was not feasible to connect the fetched dimensions to the adder trees as the used dimensions are data dependent. It is also noteworthy that the dimension-wise sparsification pushes every $\frac{\mathtt{d}}{1-\mathtt{S}}$ dimensions in the class hypervectors to retain a similar sparsity ratio of S, hence we can determine the hardware specification assuring that no more than $\frac{\mathtt{d}}{1-\mathtt{S}} \times (1-\mathtt{S}) = \mathtt{d}$ effective dimensions are generated at each cycle.

B. Associative Search

(1) Dimension-wise implementation: Fig. 4 shows the architecture of the similarity check module of SparseHD. Specifically, the flow diagram of the associative search here is shown for the non-sparse model and/or the dimension-wise sparse HD model (Fig. 4D). For the dimension-wise sparsity, the inconsequential elements across all class hypervectors are at the same indexes, hence those elements are discarded from both the classes and query hypervector, leaving a model with reduced dimensions that share a similar architecture to non-sparse HD. Each query element is multiplied by the

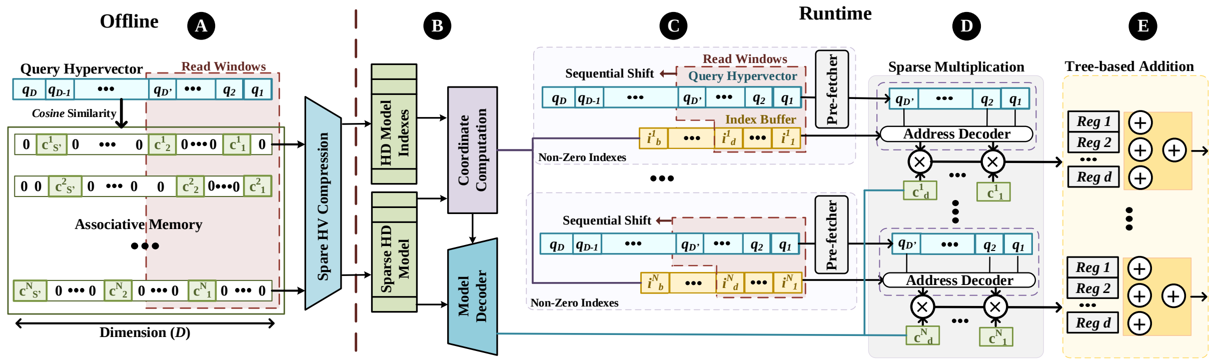


Fig. 6. FPGA implementation of the SparseHD with class-wise sparsity.

corresponding element in all class hypervectors (Fig. 4E) and are accumulated in a tree-based adder structure (Fig. 4F).

The number of input dimensions that the encoder fetches at a time, $\frac{d}{1-S} + n$ (where d is also equal to the number of dimensions processed in the associative memory), depends on the number of classes and available DSP blocks in FPGA, and the sparsity of the model. In our implementation of SparseHD on Kintex-7 FPGA KC705 Evaluation Kit with 840 DSPs, depending on the application, the value of d can also be limited by the maximum number of query elements that the encoding module generates, which is limited by LUTs count.

(2) Class-wise implementation: In an HD model with class-wise sparsity, the non-zero elements of the class hypervectors are distributed non-uniformly. To compress the sparsed HD model, we employ Compress-Sparse-Column (CSC) [35] and store the non-zero elements and their indexes in data and index vectors, respectively. The first element of the index buffer stores the number of non-zero values and the remaining elements show the number of zeros before each non-zero element. Fig. 5 demonstrates a matrix with 80% sparsity, where the five non-zero elements are stored in the data vector, and the number of zero elements between two consecutive non-zero elements proceed the total number of non-zero elements in the index vector. To compute the actual index of the non-zero elements, the coordinate computation block adds up the number of elements before the current element.

Fig. 6 elaborates the FPGA implementation of SparseHD with class-wise sparsity. The trained model is sparsified and compressed during the training (Fig. 6A). For each class, a data vector and an index vector stores the information of the HD model and the model decoder and the coordinate computation blocks are used to reconstruct the model and pass the model to the associative memory (Fig. 6B). To calculate the dot product between the query and class hypervectors, our design reads the first $\mathcal{D}'$ dimensions of the query hypervector, $\{q_{D'}, \dots, q_1\}$. These dimensions are multiplied with the first $\mathcal{D}'$ dimensions of all class hypervectors. Although query elements are stored in BRAMs, accessing them would be costly as we need to have $\mathcal{D}$ read ports. SparseHD reduces the cost of multiple read accesses by prefetching the selected $\{q_{D'}, \dots, q_1\}$ elements into a smaller distributed memory with $\mathcal{D}'$ read ports (Fig. 6C). Depending on the value of index buffer elements $\{i^1_d, \dots, i^1_1\}$, address decoder selects d query elements from the prefetched memory to multiply them with the non-zero class elements. Since each class has sparse representation with d non-zero elements, our design shifts read windows (with step d) to sequentially multiply the non-zero class elements with the corresponding elements of the query hypervector (Fig. 6D). For each class, the results of d multiplications accumulate using a tree-based adder (Fig. 6E). Each time when the read windows have been shifted over dimensions of query hypervector, the generated values are accumulated to calculate the final result of dot product for each class. Eventually, the class with the highest similarity is the result of the classification.

V. Evaluations

A. Experimental Setup

We implemented the SparseHD inference platform in Verilog and verified the timing and the functionality of the sparse models by synthesizing and mapping them using Xilinx Vivado Design Suite [36] on the Kintex-7 FPGA KC705 Evaluation Kit. We implemented SparseHD algorithmic innovation including training, model adjustment, class-wise and dimension-wise sparsity, and error estimation in C++ on CPU. We compare the SparseHD implementation with AMD Radeon R390 GPU with 8GB memory, and Intel i7 CPU with 16GB memory using the proposed sparse as well as baseline implementations. HD code of the GPU is implemented using OpenCL, while for CPU, it has has been developed in C++ and optimized for performance. We used AMD CodeXL [37] and Hioki 3334 power meter for the power measurement of the GPU and CPU, respectively. We evaluate the efficiency of the proposed SparseHD on four practical classification problems listed below:

Speech Recognition (ISOLET): the goal is to recognize voice audio of the 26 letters of the English alphabet [38].

Activity Recognition (UCIHAR): Recognizing human activity based on 3-axial linear acceleration and angular velocity captured at a constant rate of 50Hz [39].

Physical Activity Monitoring (PAMAP): Logs of eight users and three 3D accelerometers positioned on arm, chest and ankle [40]. The goal is to recognize 12 different human activities such as lying, walking, etc.

Face Detection: We exploit Caltech 10,000 web faces dataset [41]. Non-face images, are selected from CIFAR-100 and Pascal VOS 2012 datasets [42]. For the HoG feature extraction, we divide a 32x32 image to 2x2 regions for three color channels and 8x8 regions for gray-scale.

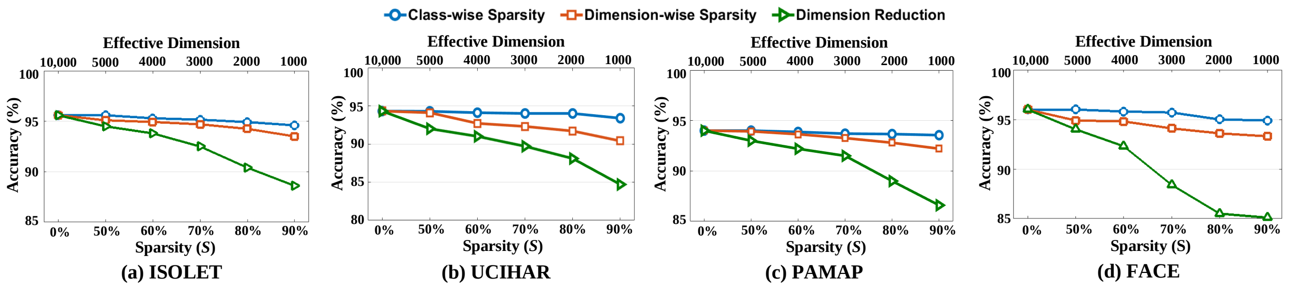


Fig. 7. Impact of sparsity on the classification accuracy of the class-wise and dimension-wise sparse models (the blue and orange curves). The green curves correspond to non-sparse models with smaller dimensionality such that the number of dimensions matches the number of non-zeros in the sparse hypervectors.

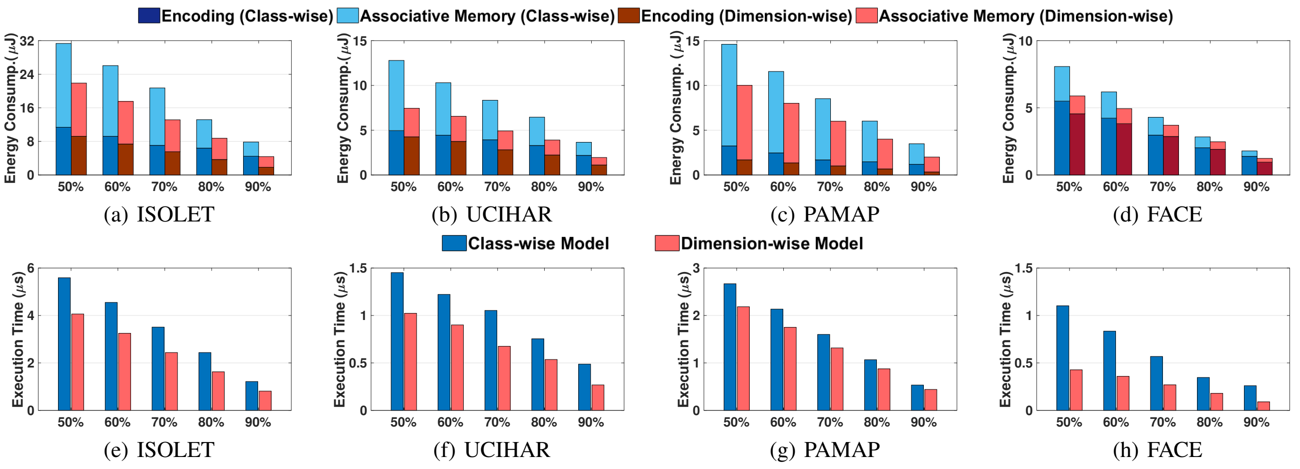


Fig. 8. Energy consumption and execution time of FPGA-based implementation of SparseHD with class-wise/dimension-wise models in different sparsity.

B. SparseHD *Accuracy-Efficiency*

(1) Accuracy versus sparsity: Fig. 7 shows the classification accuracy of the baseline HD (0% sparsity) and SparseHD as the model sparsity increases from 50% to 90%. SparseHD with both dimension-wise and class-wise models has very stable accuracy when the model sparsity scales up to 90%, albeit applications have different sensitivity to sparsity. The results also show that at the same level of sparsity, the class-wise model provides higher accuracy as compared to the dimension-wise model, i.e., the class-wise can work in higher sparsity while providing the same accuracy as dimension-wise model. It stems from the fact that the class-wise model exploits all dimensions of the hypervectors to represent the class pattern, while the dimension-wise model reduces dimensionality by discarding the entire dimensions for all existing classes, resulting in lower flexibility during the retraining process.

For a fair evaluation, Fig. 7 also compares the accuracy of SparseHD with a non-sparse but low-dimensional model which has been trained with the same effective dimension as sparse models (and the same number of total training iterations). Evidently, the non-sparse low-dimensional HD provides lower accuracy than the sparse model using the same effective dimensions. This lower accuracy comes from the fundamental concept behind HD which requires the model to be built upon the nearly orthogonal base hypervectors. However, the mathematics governing the high dimensional space do not perfectly work when the hypervector dimensionality is reduced. SparseHD only discards the inconsequential model elements and still maintains the orthogonality of the hypervectors in the high dimensional space.

(2) Energy efficiency versus sparsity: Fig. 8(a)-(d) show the energy breakdown of SparseHD, mapped four different applications on FPGA, using both dimension-wise and class-wise sparsification. The encoding module takes different ratios of the total energy consumption depending on the number of features and classes. Its energy consumption improves with the sparsity of model as higher sparsity decreases the number of effective dimensions and thus reduces the number of query elements which the encoding module needs to generate. For all applications, the class-wise model consumes higher encoding energy as compared to the dimension-wise model. In the class-wise model, usually fewer dimensions are zero across all class hypervectors, while in dimension-wise sparse model the number of zero dimensions across all classes increases linearly with the model sparsity. Based on the results, SparseHD with 90% sparsity, on average, reduces the effective number of query elements to 10% and 48% for dimension-wise and class-wise sparse models, which results in 9.5× and 4.4× higher energy efficiency compared to baseline HD encoding module.

Sparsity also improves the energy efficiency of associative memory for both class-wise and dimension-wise sparse models. Similar to encoding, at the same level of sparsity, the class-wise SparseHD provides lower efficiency than dimension-wise model. This is because, in class-wise model, the non-zero elements are distributed in all $\mathcal{D}$ dimensions of a hypervector, hence FPGA needs a large amount of sequential memory reads to perform all sparse multiplications between a query and class hypervectors. This incurs the overhead of fetching and storing

TABLE I
NORMALIZED ENERGY-DELAY PRODUCT (EDP) IMPROVEMENT OF APPLICATIONS ENSURING DIFFERENT QUALITY LOSS.

Quality Loss		**0%**	**0.3%**	**0.5%**	**1%**	**1.5%**	**2%**
ISOLET	*Dimension-wise*	1×	1×	1×	22.6×	51.0×	51.0×
	Class-wise	4.1×	6.1×	9.9×	22.6×	76.1×	76.1×
UCIHAR	*Dimension-wise*	1×	6.8×	3.0×	3.0×	3.0×	7.8×
	Class-wise	1×	4.2×	10.7×	10.7×	29.4×	29.4×
PAMAP	*Dimension-wise*	1×	3.9×	3.9	11.1×	24.7×	24.7×
	Class-wise	3.3×	5.3×	69.5×	69.5×	69.5×	69.5×
FACE	*Dimension-wise*	1×	1×	1×	1×	2.5×	3.3×
	Class-wise	1.2×	4.5×	4.5×	11.3×	23.7×	23.7×
AVERAGE	*Dimension-wise*	1×	3.2×	3.2×	10.4×	22.3×	23.9×
	Class-wise	2.4×	5.0×	11.4×	28.5×	49.7×	49.7×

more dimensions, resulting in lower computation efficiency. In contrast, dimension-wise model reduces the hypervector dimensions, and the corresponding hardware does not have the overhead of reading non-zero dimensions.

(3) Performance versus sparsity: Fig. 8 (e)-(h) show the execution time of the SparseHD using dimension-wise and class-wise models. SparseHD is implemented in pipelined stages such that the delay of encoding module is masked by the execution time of the associative memory. For each application, we fully utilized the FPGA resources to maximizes the performance. Our evaluation shows that for both sparse models, SparseHD performance improves by increasing the sparsity of the class hypervectors. The execution time of SparseHD is limited by the minimum encoding or associative memory throughput. As explained already, the maximum number of query elements that SparseHD can process d at a time depends on feature size and number of classes. For SparseHD with a large number of features, the encoding module becomes the bottleneck, while for SparseHD with a large number of classes the associative memory limits d. Comparing the class-wise and dimension-wise models also shows that dimension-wise associative memory mostly utilizes DSPs, while using less LUTs than the class-wise model. This enables the dimension-wise model to use the majority of FPGA LUTs for encoding module, resulting higher throughput.

(4) Accuracy-efficiency trade-off: Table I lists the normalized energy-delay product (EDP) improvement of SparseHD using dimension-wise and class-wise models while ensuring different quality loss bound. The EDP results are relative to the FPGA-based implementation of the baseline non-sparse HD code. Although at the same *sparsity* level the class-wise model is less efficient than the dimension-wise model, it provides higher efficiency than at the same level of *accuracy*. This is because of the higher tolerance of the class-wise model to sparsity, which enables it to work with higher sparsity compared to dimension-wise model. For example, when SparseHD ensures less than 0.5% quality loss ($\Delta E = 0.5\%$), the dimension-wise and class-wise models provide 3.2× and 11.4× EDP improvement compared to the baseline HD model running on FPGA. Similarly, ensuring the quality loss of less than 1% and 1.5%, SparseHD with the class-wise model achieves 28.5× and 49.7× EDP improvement as compared to the FPGA implementation of baseline HD.

C. HD Acceleration on Different Platforms

Fig. 9 shows the energy consumption and execution time of HD computing applications running on different platforms described before. All platforms run the baseline HD code with $\mathcal{D} = 10,000$ dimensions (non-sparse model) and the sparse model with 50% and 90% sparsity. The results are normalized to GPU running non-sparse HD algorithm. Accordingly, FPGA provides on average 8.7× (18.3×) lower energy consumption and 1.9× (178.4×) faster computation compared to the GPU (CPU) when running HD in full dimension. The higher efficiency of the FPGA rises from the fine-grained pipeline and parallelism and granted flexibility to manage the irregular data patterns in fetched data. Sparsity improves the efficiency of both GPU and FPGA platforms, however, the improvement is more significant on FPGA. For example, GPU running 90% class-wise (dimension-wise) sparse model provides maximum 1.3× and 1.4× (3.5× and 3.3×) speedup and energy efficiency improvement compared to the GPU running a non-sparse model. However, FPGA running the class-wise (dimension-wise) model with the same sparsity achieves 15.0× 48.5× (19.7× 84.1×) speedup and energy efficiency as compared to the GPU, respectively.

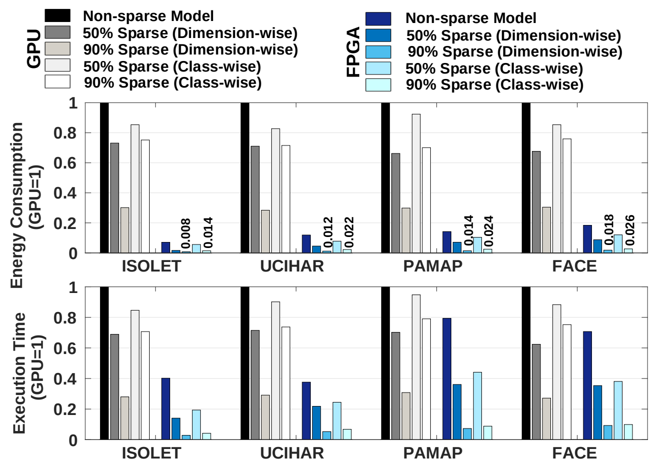


Fig. 9. Energy consumption and execution time of the baseline HD on GPU and FPGA platforms

VI. CONCLUSION

In this paper, we proposed a novel algorithm-architecture platform, SparseHD, for efficient Hyperdimensional computing, as a new paradigm in learning applications. The algorithmic innovation of SparseHD introduces different concepts of sparsity to the representative class hypervectors, which, consequently, reduce the computations required for HD inference, leading to more effective utilization of available resources. We also proposed an FPGA implementation of SparseHD which enables efficient realization of sparsity in the hardware level granted by the bit-level parallelism and pipelining supported by FPGA. We conducted an extensive set of experiments using different benchmarks, sparsity rates, and hardware platforms to evaluate the proposed framework.

ACKNOWLEDGEMENTS

This work was partially supported by CRISP, one of six centers in JUMP, an SRC program sponsored by DARPA, Office of Naval Research (N00014-17-1-2500), and also NSF grants #1730158, #1527034, and #1619261.

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. IEEE, 2015, pp. 4705–4713.

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

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

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

[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*, 2016, pp. 770–778.

[6] A. Zaslavsky, C. Perera, and D. Georgakopoulos, "Sensing as a service and big data," *arXiv preprint arXiv:1301.0159*, 2013.

[7] R. Khan, S. U. Khan, R. Zaheer, and S. Khan, "Future internet: the internet of things architecture, possible applications and key challenges," in *Frontiers of Information Technology (FIT), 2012 10th International Conference on*. IEEE, 2012, pp. 257–260.

[8] Y. Sun, H. Song, A. J. Jara, and R. Bie, "Internet of things and big data analytics for smart and connected communities," *IEEE Access*, vol. 4, pp. 766–773, 2016.

[9] 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.

[10] M. Imani *et al.*, "A framework for collaborative learning in secure high-dimensional space," in *Cloud Computing (CLOUD)*. IEEE, 2019, pp. 1–6.

[11] O. Rasanen 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.

[12] O. J. Räsänen and J. P. 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. 27, no. 9, pp. 1878–1889, 2016.

[13] A. Rahimi, S. Datta, D. Kleyko, E. P. Frady, B. Olshausen, P. Kanerva *et al.*, "High-dimensional computing as a nanoscalable paradigm," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 64, no. 9, pp. 2508–2521, 2017.

[14] M. Imani, D. Kong, A. Rahimi, and T. Rosing, "Voicehd: Hyperdimensional computing for efficient speech recognition," in *Rebooting Computing (ICRC), 2017 IEEE International Conference on*. IEEE, 2017, pp. 1–8.

[15] M. Imani *et al.*, "Bric: Locality-based encoding for energy-efficient brain-inspired hyperdimensional computing," in *ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2019, pp. 1–6.

[16] M. Ghasemzadeh, M. Samragh, and F. Koushanfar, "Rebnet: Residual binarized neural network," in *2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 2018, pp. 57–64.

[17] Y. Umuroglu, N. J. Fraser, G. Gambardella, M. Blott, P. Leong, M. Jahre *et al.*, "Finn: A framework for fast, scalable binarized neural network inference," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 2017, pp. 65–74.

[18] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," in *European Conference on Computer Vision*. Springer, 2016, pp. 525–542.

[19] M. S. Riazi, M. Samragh, H. Chen, K. Laine, K. Lauter, and F. Koushanfar, "Xonn: Xnor-based oblivious deep neural network inference," *arXiv preprint arXiv:1902.07342*, 2019.

[20] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow *et al.*, "Intriguing properties of neural networks," *arXiv preprint arXiv:1312.6199*, 2013.

[21] B. D. Rouhani, M. Samragh, M. Javaheripi, T. Javidi, and F. Koushanfar, "Deepfense: Online accelerated defense against adversarial deep learning," in *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2018, pp. 1–8.

[22] 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*, 2010, pp. 2–6.

[23] A. Joshi, J. T. Halseth, and P. Kanerva, "Language geometry using random indexing," in *International Symposium on Quantum Interaction*. Springer, 2016, pp. 265–274.

[24] M. Imani *et al.*, "Hdna: Energy-efficient dna sequencing using hyperdimensional computing," in *IEEE BHI*. IEEE, 2018, pp. 271–274.

[25] Y. Kim *et al.*, "Efficient human activity recognition using hyperdimensional computing," in *IoT*. ACM, 2018, p. 38.

[26] M. Imani *et al.*, "Hdcluster: An accurate clustering using brain-inspired high-dimensional computing," in *DATE*. IEEE/ACM, 2019.

[27] 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*. IEEE, 2017, pp. 445–456.

[28] M. Imani *et al.*, "A binary learning framework for hyperdimensional computing," in *DATE*. IEEE/ACM, 2019.

[29] S. Gupta *et al.*, "Felix: Fast and energy-efficient logic in memory," in *IEEE/ACM ICCAD*. IEEE, 2018, pp. 1–7.

[30] A. DeHon, "The density advantage of configurable computing," *Computer*, vol. 33, no. 4, pp. 41–49, 2000.

[31] B. Logan *et al.*, "Mel frequency cepstral coefficients for music modeling." in *ISMIR*, vol. 270, 2000, pp. 1–11.

[32] 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*. ACM, 2016, pp. 64–69.

[33] M. Schmuck, L. Benini, and A. Rahimi, "Hardware optimizations of dense binary hyperdimensional computing: Rematerialization of hypervectors, binarized bundling, and combinational associative memory," *arXiv preprint arXiv:1807.08583*, 2018.

[34] M. Imani *et al.*, "Fach: Fpga-based acceleration of hyperdimensional computing by reducing computational complexity," in *ASPDAC*. ACM, 2019, pp. 493–498.

[35] L. Lu and Y. Liang, "Spwa: An efficient sparse winograd convolutional neural networks accelerator on fpgas," in *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*. IEEE, 2018, pp. 1–6.

[36] T. Feist, "Vivado design suite," *White Paper*, vol. 5, 2012.

[37] "Amd," http://developer.amd.com/tools-and-sdks/opencl-zone/codexl/.

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

[39] "Uci machine learning repository," https://archive.ics.uci.edu/ml/datasets/Daily+and+Sports+Activities.

[40] A. Reiss and D. Stricker, "Creating and benchmarking a new dataset for physical activity monitoring," in *Proceedings of the 5th International Conference on PErvasive Technologies Related to Assistive Environments*. ACM, 2012, p. 40.

[41] G. Griffin, A. Holub, and P. Perona, "Caltech-256 object category dataset," 2007.

[42] M. Everingham, S. A. Eslami, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes challenge: A retrospective," *International journal of computer vision*, vol. 111, no. 1, pp. 98–136, 2015.