Efficient Associative Search in Brain-Inspired Hyperdimensional Computing

Mohsen Imani, Justin Morris, Helen Shu, Shou Li, and Tajana Rosing
Department of Computer Science and Engineering, University of California at San Diego
{moimani, j1morris, hsshu, shl297, tajana}@ucsd.edu

***Abstract*—Brain-inspired Hyperdimensional (HD) computing exploits hypervector operations, such as cosine similarity, to perform cognitive tasks. In inference, cosine similarity involves a large number of computations which grows with the number of classes, this results in significant overhead. In this paper, we propose a grouping approach that reduces inference computations by checking a subset of classes during inference. In addition, we propose a quantization approach which removes costly multiplications by using the power of two weights. We also remove computations by caching hypervector magnitudes to reduce cosine similarity operations to dot products. Our proposed design achieves 11.6× energy efficiency and 8.3× speedup as compared to the baseline HD design.**

I. Introduction

The emergence of the Internet of Things has created an abundance of small embedded devices. Many of these devices may be used for cognitive tasks such as: face detection, speech recognition, image classification, activity monitoring, etc. Learning algorithms such as Deep Neural Networks (DNNs) give accurate results for cognitive tasks [1]. However, these embedded devices have limited resources and cannot run such resource-intensive algorithms. Instead, many of them send the data they collect to the cloud server, for feature analysis. However, this is not desirable due to privacy concerns, security concerns, and network resources. Thus, we need more efficient light-weight classifiers in order to perform such cognitive tasks on the embedded systems.

Brain-inspired Hyperdimensional (HD) computing can be used as a light-weight classifier to perform cognitive tasks on resource-limited systems [2]. HD computing is modeled after how the brain works, using patterns of neural activity instead of computational arithmetic. Past research utilized high-dimension vectors (D $\geq$ 10,000) called hypervectors, to represent neural patterns. It showed that HD computing is capable of providing high accuracy results for a variety of tasks such as: language recognition, face detection, speech recognition, classification of time-series signals, and clustering [3], [4], [5], [6], [7]. Results are obtained at a much lower computational cost as compared to other learning algorithms.

HD computing performs the classification task after encoding all data points to high-dimensional space. The HD training happens by linearly combining the encoded hypervectors and creating a hypervector representing each class. In inference, HD uses the same encoding module to map a test data point to high-dimensional space. Then the classification task checks the similarity of the encoded test hypervector with all pre-trained class hypervectors. This similarity check is the main HD computation during the inference. Often done with a cosine, which involves a large number of costly multiplications that grows with the number of classes [8]. Given an application with k classes, inference requires $k * D$ additions and multiplications to perform, where D is the hypervector dimension. Thus, this similarity check can be costly for embedded devices with limited resources.

In this paper, we propose a robust and efficient solution to reduce the computational complexity and cost of HD computing while maintaining comparable accuracy. The proposed HD framework exploits the mathematics in high dimensional space in order to limit the number of classes checked upon inference, thus reducing the number of computations needed for query requests. We add a new layer before the primary HD layer to decide which subset of class hypervectors should be checked as possible classes for the output class. This reduces the number of additions and multiplications needed for inference. In addition, our framework removes the costly multiplication from the similarity check by quantizing the values in the trained HD model with power of two values. Our approach integrates quantization with the training process in order to adapt the HD model to work with the quantized values. We have evaluated our proposed approach on three practical classification problems. Our evaluations show that the proposed design is 11.6× more energy efficient and 8.3× faster as compared to the baseline HD while providing similar classification accuracy.

II. Hyperdimensional Computing

HD computing uses long vectors with dimensionality in the thousands [2]. There are many nearly orthogonal vectors in high-dimensional space. HD combines these hypervectors with well-defined vector operations while preserving most of their information. No component has more responsibility to store any piece of information than any other component because hypervectors are holographic and (pseudo) random with i.i.d. components and a full holistic representation. The mathematics governing the high-dimensional space computations enables HD to be easily applied to many different learning problems.

Figure 1 shows an overview of the structure of the HD model. HD consists of an encoder, trainer, and associative search block. The encoder maps data points into high-dimensional space. These hypervectors are then combined in a trainer block to form class hypervectors, which are then stored

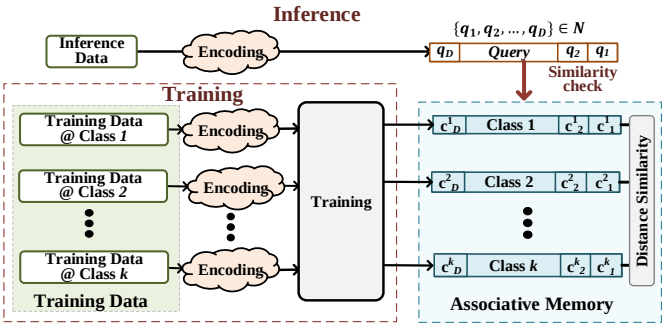

Fig. 1. Overview of HD computing performing classification task.

in an associative search block. In inference, an input test data is encoded to high-dimensional space using the same encoder as the training module. The classifier uses cosine similarity to check the similarity of the encoded hypervector with all class hypervectors and find the most similar one.

A. Encoding

Consider a feature vector $\mathbf{v} = \langle v_1, \ldots v_n \rangle$. The encoding module takes this n-dimensional vector and converts it into a D-dimensional hypervector ($D >> n$). The encoding is performed in three steps, which we describe below.

We use a set of pre-computed *level* or base hypervectors to consider the impact of each feature value [9]. To create these level hypervectors, we compute the minimum and maximum feature values among all data points, $\mathbf{v}_{min}$ and $\mathbf{v}_{max}$, then quantize the range of $[\mathbf{v}_{min}, \mathbf{v}_{max}]$ into Q levels, $\mathbf{L} = \{\mathbf{L}_1, \cdots, \mathbf{L}_Q\}$. Each of these quantized scalars corresponds to a D-dimensional hypervector [9].

Once the base hypervectors are generated, each of the n elements of the vector v are independently quantized and mapped to one of the base hypervectors. The result of this step is n different binary hypervectors, each of which is D-dimensional. In the last step, the n (binary) hypervectors are combined into a single D-dimensional (non-binary) hypervector. To differentiate the impact of each feature index, we devise *ID* hypervectors, $\{\mathbf{ID}_1, \cdots, \mathbf{ID}_n\}$. An **ID** hypervector has the binarized dimensions, i.e., $\mathbf{ID}_i \in \{0,1\}^D$. We create **ID**s with random binary values so that the *ID* hypervectors of different feature indexes are nearly orthogonal:

$$\delta(\mathbf{ID}_i, \mathbf{ID}_j) \simeq D/2 \qquad (i \neq j \ \& \ 0 < i, j \leqslant n)$$

where the similarity metric, $\delta(\mathbf{ID}_i, \mathbf{ID}_j)$, is the Hamming distance between the two **ID** hypervectors.

The orthogonality of *ID* hypervectors is ensured as long as the hypervector dimensionality is large enough compared to the number of features in the original data point ($D >> n$). As Figure 1 shows, the aggregation of the n binary hypervectors is computed as follows:

$$\mathbf{H} \ = \ \mathbf{ID}_1 \oplus \overline{\mathbf{L}}_1 \ + \ \mathbf{ID}_2 \oplus \ \overline{\mathbf{L}}_2 + \ldots \ + \ \mathbf{ID}_n \oplus \overline{\mathbf{L}}_n.$$

where, $\oplus$ is XOR operation, $\mathbf{H}$ is the aggregation, and $\overline{\mathbf{L}}_i$ is the binary hypervector corresponding to the $i-th$ feature of vector $\mathbf{v}$.

B. Classification in HD Space

In HD, training is performed in high-dimensional space by element-wise addition of all encoded hypervectors in each existing class. The result of training will be k hypervectors with D dimensions, where k is the number of classes. For example, the i^{th} class hypervector can be computed as: $\mathbf{C^i} = \sum_{\forall j \in class_i} \mathbf{H_j}$

In inference, HD uses encoding and associative search for classification. First, HD uses the same encoding module as the training module to map a test data point to a *query hypervector*. In HD space, the classification task then checks the similarity of the query with all class hypervectors. The class with the highest similarity to the query is selected as the output class. Since in HD information is stored as the pattern of values, the cosine is a suitable metric for similarity check.

C. HD Computing Challenges

To understand the main bottleneck of HD computing during inference, we evaluate the HD inference on three practical classification applications, including: speech recognition [10], activity recognition [11], and image recognition [12]. All evaluations are performed on Intel i7 7600 CPU with 16GB memory. Our results show that the associative search takes about 83% of the total inference execution. This is because the cosine similarity has many multiplications between a query and the class hypervectors. Prior work tried to binarize the HD model after the training [3]. This method simplifies the cosine similarity to Hamming distance measurement, which can run faster and more efficiently in hardware. However, our evaluation of three practical applications shows that binarization reduces the classification accuracy of HD with integer model by 11.4% on average. This large accuracy drop forces HD to use cosine similarity which has a significant cost when running on embedded devices.

III. HD Computing Acceleration

In this paper, we propose three optimization methods that reduce the cost of associative search during inference by at least an order of magnitude. Figure 2 shows the overview of the proposed optimizations. The first approach simplifies the cosine similarity calculations to dot products between the query and class hypervectors. The second reduces the number of required operations in the associative search by adding a category layer to HD computing that decides what subset of class hypervectors needs to be checked for the output class. The third removes the costly multiplications from the similarity check by quantizing the HD model after training. In the following subsections, we explain the details of each proposed approach.

A. Similarity Check: Cosine or Dot Product?

During inference, HD computation encodes input data to a query hypervector, $\mathbf{H} = \{h_D, \ldots, h_2, h_1\}$. Associative memory then measures the cosine similarity of this query with k stored class hypervectors $\{\mathbf{C}^1, \ldots, \mathbf{C}^k\}$, where $\mathbf{C}^i = \{c^i_D, \ldots, c^i_2, c^i_1\}$ is the class hypervector corresponding to the i^{th} class (Figure 2a). The cosine similarity can be expressed as $\delta = \mathbf{H} \cdot \mathbf{C}^i / |\mathbf{H}||\mathbf{C}^i|$,

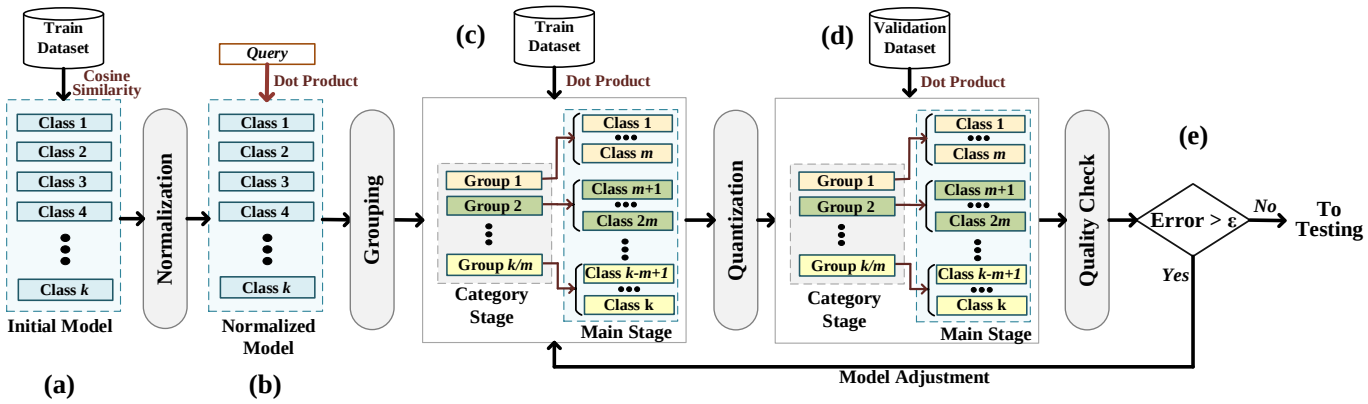

Fig. 2. Overview of proposed optimization approaches to improve the efficiency of associative search.

where $\mathbf{H} \cdot \mathbf{C}^i$ indicates the dot product between the hypervectors, and $|\mathbf{H}| = \mathbf{H} \cdot \mathbf{H}$ and $|\mathbf{C}^i| = \mathbf{C}^i \cdot \mathbf{C}^i$ show the magnitudes of the query and class hypervector. However, it is very expensive to calculate the operand magnitude every time. During the similarity check, the query hypervector is common between all classes. Thus, we can skip the calculation of the query magnitude, since the goal of HD is to find the maximum relative similarity, not the exact cosine values. On the other hand, as Figure 2 b shows, the magnitude of each class hypervector can be computed once after the training. Therefore, the associative search can store the normalized class hypervectors ($\mathbf{C}^i/|\mathbf{C}^i|$ for $i \in 1, \ldots, k$). This speeds up the similarity at inference by about 3 times as compared to cosine.

B. Two level search

Although the dot product reduces the cost of the associative search, this similarity check still involves many computations. For example, for an application with k classes, associative search computes $k \times D$ multiplication/addition operations, where D is the hypervector dimension. In addition, in existing HD computing approaches [9], [3], the cost of associative search increases linearly with the number of classes. For example, speech recognition with $k = 26$ classes has $13\times$ more computations and $5.2\times$ slower inference as compared to face detection with $k = 2$ classes. Since embedded devices often do not have enough memory and computing resources, processing HD applications with large numbers of classes are more inefficient.

We propose a method which has two layer classification: category and main stages. Figure 2c shows the overview of the proposed approach. First, we group the trained class hypervectors into k/m categories based on their similarity, where k and m are the number of classes and group size respectively. For example, $m = 2$ indicates that we group every two class hypervectors into a single hypervector. Next, we build a new HD model, called category stage, which stores all k/m group hypervectors. Instead of searching k hypervectors to classify a data point, we first search in the category stage to identify a group of classes that the query belongs to (among k/m group hypervectors). Afterwards, we continue the search in the main HD stage but only with the class hypervectors corresponding to the selected group.

C. Quantization

Although grouping the class hypervectors reduces the number of computations, the dot product similarity check still has many costly multiplications. In this work, we propose a method which removes the majority of the multiplications from the HD similarity check. After training the HD model, we quantize each class value to the closest power of two (2^i, $i \in \mathbb{Z}$). This eliminates the multiplication by allowing bit shift operations. However, this quantization is not error free. For example, the closest power of two for the number 46 would be either 32 or 64. Both of these numbers are far from the actual value. This approximation can add large errors to HD computing. The amount of error depends on the number of classes and how similar the classes are. For applications with many classes or highly correlated class hypervectors, this quantization can have a large impact of the classification accuracy.

In this work, we look at the possibility of performing a more precise but lower power quantization approach (Figure 2d). Our method assigns each class element to a combination of 2 power of two values ($2^i + 2^j$, i & $j \in \mathbb{Z}$). This quantization can assign each trained class element to a value which is much closer to the actual element. For example, using this approach the number 46 can be assigned to $48 = 2^5 + 2^4$, which is very close to the actual value. This enables more precise quantization with correspondingly lower impact on accuracy. This strategy implements multiplication using two shifts and a single add operation, which is still faster and more efficient than the actual multiplication. After training the HD model, we assign the class elements in both category and main stage to the closest quantized value.

Since class hypervectors are highly correlated, even this quantization can have a large impact on the classification accuracy. Quantization introduces this quality loss because the HD model is not trained to work with the quantized values. In order to ensure a minimum quality loss, we integrate quantization with the HD model retraining. This enables the HD model to learn how to work with quantized values. In the algorithm explained in Section III-D, after getting a new adjusted model, we quantize all hypervector values in the category and main stage. This approach reduces the possible quality loss due to quantization. In Section IV-B, we discuss

TABLE I
CLASSIFICATION ACCURACY OF HD WITH INTEGER (BASELINE) AND QUANTIZED MODEL.

	Integer Model				Quantized (2^i)				Quantized (2^i+2^j)			
# of Classes in a Group	**1**	**2**	**3**	**4**	**1**	**2**	**3**	**4**	**1**	**2**	**3**	**4**
Speech Recognition	91.72%	88.39%	88.01%	87.04%	91.73%	83.00%	85.70%	83.39%	90.64%	88.45%	88.26%	88.39%
Activity Recognition	95.50%	95.49%	95.37%	94.98%	93.50%	92.73%	94.92%	94.72%	94.27%	94.34%	95.62%	94.72%
Image Recognition	92.97%	93.25%	91.02%	89.11%	89.99%	92.54%	91.05%	80.62%	91.51%	93.00%	92.28%	89.31%

the impact of quantization on HD classification accuracy.

D. Training & Inference in Proposed Design

Training Figure 2 shows the training process of HD with grouped hypervectors. We first train a normal HD computing model, where each hypervector represents an existing class (Figure 2a). Next, we normalize the class hypervectors (Figure 2b) and then check the similarity of the trained class hypervectors in order to group the classes. In our approach, we group every m class hypervectors into a single hypervector, so in the end, we have k/m group hypervectors (Figure 2c). The grouping happens by checking the similarity of class hypervectors in pairs and merging classes with the highest similarity. The selected class hypervectors are added together to generate a group hypervector. We store these k/m group hypervectors into the category stage. After that, we quantize the values of the grouped model (Figure 2d). This one-shot trained model can be used to perform the classification task at inference.

Model Adjustment In order to get better classification accuracy, we can adjust the HD model with the training dataset for a few iterations (Figure 2e). The model adjustment starts in the main HD stage. During a single iteration, HD checks the similarity of all training data points, say $\mathbf{H}$, with the current HD model. If data is wrongly classified by the model, HD updates the model by (i) adding the data hypervector to a class that it belongs to, and (ii) subtracting it from a class which it was wrongly matched with:

$$\textbf{Main}\begin{cases}\widetilde{\mathbf{C}}^{c}_{main}=\mathbf{C}^{c}_{main}+\mathbf{H}, & \text{where } \mathbf{C}^{c}_{main} \textit{ is correct}\\ \widetilde{\mathbf{C}}^{w}_{main}=\mathbf{C}^{w}_{main}-\mathbf{H}, & \text{where } \mathbf{C}^{w}_{main} \textit{ is wrong}\end{cases}$$

We similarly update two corresponding hypervectors in the category stage by adding and subtracting the query hypervector:

$$\textbf{Category}\begin{cases}\widetilde{\mathbf{C}}^{c}_{category}=\mathbf{C}^{c}_{category}+\mathbf{H}, & \text{where } \mathbf{C}^{c}_{main}\in\mathbf{C}^{c}_{category}\\ \widetilde{\mathbf{C}}^{w}_{category}=\mathbf{C}^{w}_{category}-\mathbf{H}, & \text{where } \mathbf{C}^{w}_{main}\in\mathbf{C}^{w}_{category}\end{cases}$$

The model adjustment needs to be continued for a few iterations until the HD accuracy stabilizes over the validation data, which is a part of the training dataset. After training and adjusting the model offline, it can be loaded onto embedded devices to be used for inference.

Inference The proposed approach works very similarly to the baseline HD computing, except now there are two stages. First, we check the similarity of a query hypervector in the category stage. A category hypervector with the highest cosine similarity is selected to continue the search in the main stage. Here, we check the similarity of the query hypervector against the classes within the selected category. For example, in Figure 2c, if group 2 had the highest cosine similarity with the query hypervector, then only the green class hypervectors are selected for search in the main stage. Finally, a class with the highest cosine similarity in the main stage is selected as the output class. This approach reduces the number of required operations. For an application with k classes, our approach reduces the number of required similarity checks from k to $k/m+m$ hypervectors. For example, for an application with $k=16$ and $m=4$, the number of required operations is reduced by a factor of 2.

IV. EVALUATION

A. Experimental Setup

We perform HD training and retraining using C++ implementation on Intel Core i7 processor with 16 GB memory (4-core, 2.8GHz). We describe the inference functionality using RTL System-Verilog and use standard digital ASIC flow to design dedicated hardware. For the synthesis, we use *Synopsys Design Compiler* with the TSMC 45 nm technology library, the general purpose processor with high V_{TH} cells. We extract the design switching activity using *ModelSim* and measured the power consumption of HD designs using *Synopsys PrimeTime* at (1 V, 25°C, TT) corner.

We test the efficiency of the proposed approach on three practical applications:

Speech Recognition (ISOLET) [10]: Recognize voice audio of the 26 letters of the English alphabet. The training and testing datasets are taken from the Isolet dataset. This dataset consists of 150 subjects speaking each letter of the alphabet twice. The speakers are grouped into sets of 30 speakers. The training of hypervectors is performed on *Isolet* 1,2,3,4, and tested on *Isolet* 5.

Activity Recognition (UCIHAR) [11]: Detect human activity based on 3-axial linear acceleration and 3-axial angular velocity that has been captured at a constant rate of 50Hz. The training and testing datasets are taken from the Human Activity Recognition dataset. This dataset contains 10,299 samples each with 561 attributes.

Image Recognition (IMAGE) [12]: Recognize hand-written digits 0 through 9. The training and testing datasets are taken from the Pen-Based Recognition of Handwritten Digits dataset. This dataset consists of 44 subjects writing each numerical digit 250 times. The samples from 30 subjects are used for training and the other 14 are used for testing.

B. Accuracy

In this section, we study the impact of quantization and two level search classification accuracy. Table I shows the HD classification accuracy for four different configurations when we categorize the class hypervectors into groups of $m=1$ to 4. The configuration $m=1$ is the baseline HD, where we do not have any grouping. HD in $m=2$ configuration is where each group consists of two class hypervectors. This generates k/m hypervectors in category stage. Our evaluation shows that grouping has a minor impact on the classification accuracy (0.6% on average). HD classification accuracy is also a weak function of grouping configurations. However, the number of hypervectors in the category stage affects the number of computations needed for inference. Therefore, we choose the grouping approach that minimizes the number of required operations for a given application. For example, for activity recognition with $k=12$ classes, the grouping with $m=4$ results in maximum efficiency, since it reduces the number of effective hypervectors from $k=12$ to $k/m+m=7$.

Table I also shows the HD classification accuracy for two types of quantization. Our results show that HD on average loses 3.7% in accuracy when quantizing the trained model values to power of two values (2^i). However, quantizing the values to 2^i+2^j values enables HD to provide similar accuracy to HD with integers with less than 0.5% error. This quantization results in 2.2× energy efficiency improvement and 1.6× speedup by modeling the multiplication with two shifts and a single add operation.

C. Efficiency

The goal is to have HD be small and scalable so that is can be stored and processed on embedded devices with limited resources. In the conventional HD, each class is represented using a single hypervector. We address this issue by grouping classes together, which significantly lowers the number of computations, and with quantization, which removes costly multiplications from the similarity check.

Figure 3 compares the energy consumption and execution time of the proposed approach with the baseline HD computing during inference. We reported results such that reader can see the impact of different optimizations. To have fair comparison, the baseline HD uses the same encoding and number of retraining iterations as proposed design. Our evaluation shows that grouping of class hypervectors can achieve on average 5.3× energy efficiency improvement and 4.9× faster as compared to the baseline HD using cosine similarity. In addition, quantization (2^i+2^j) of class elements can further improve the HD efficiency by removing costly multiplications. Our evaluations show that HD enhancing with both grouping and quantization achieves 11.6× energy efficiency and 8.3× speedup as compared to baseline HD using cosine while providing similar classification accuracy.

V. Conclusion

Hyperdimensional computing is a promising solution to perform light-weight classification tasks, however, HD is computationally expensive when working with applications that have large numbers of classes. In this paper, we propose three novel approaches to reduce the HD computation cost during inference. Our first approach simplifies cosine similarity operations to dot product operations by caching class hypervector magnitudes. Our second approach reduces the number computation during inference by grouping the class hypervectors and performing similarity check in two stages. Our third approach quantizes the HD trained model and removes costly multiplications from the similarity check. Using the proposed approach enables us to exploit HD as light-weight classifier for computing on the edge, such as on small embedded devices.

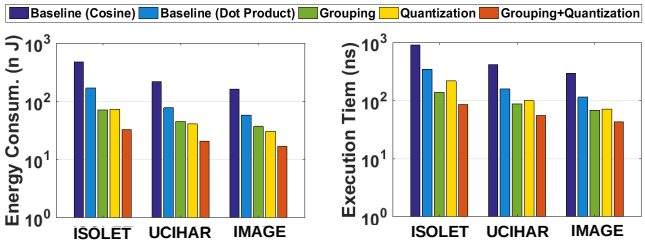

Fig. 3. Energy consumption and execution time of HD using proposed optimization approaches.

Acknowledgements

This work was partially supported by CRISP, one of six centers in JUMP, an SRC program sponsored by DARPA. NSF grants #1730158, #1527034, and CNS-1339335.

References

[1] 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.

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

[3] 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.

[4] M. Imani, J. Hwang, T. Rosing, A. Rahimi, and J. M. Rabaey, "Low-power sparse hyperdimensional encoder for language recognition," *IEEE Design & Test*, vol. 34, no. 6, pp. 94–101, 2017.

[5] 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.

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

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

[8] 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.

[9] 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.

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

[11] 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.

[12] "Uci machine learning repository." https://archive.ics.uci.edu/ml/datasets/Pen-Based+Recognition+of+Handwritten+Digits.