A Framework for Efficient and Binary Clustering in High-Dimensional Space

***Abstract*—Name of Design should be update. Text need review! Algorithms need to be smaller in size, should fit into 6pages! Today's applications generate a large amount of data where the majority of the data are not associated with any labels. Clustering methods are the most commonly used algorithms for data analysis. However, running clustering algorithms on embedded devices are significantly slow as the computation involves large amount of complex pairwise similarity measurements. In this paper, we proposed Design, an adaptive and fully binary clustering in high-dimensional space. Instead of using complex metrics, e.g., Euclidean distance, to perform similarity, Design introduces a non-linear encoder to map data points into sparse high-dimensional space. Design encoder simplifies the similarity search, the most costly/frequent clustering operation, to Hamming distance which can be accelerated in today's hardware. Design performs clustering by assigning each data point to a set of initialized centers. Then, it updates the centers adaptively based on: (i) data points assigned to each cluster, and (ii) how confident they have been match with each center. This adaptive cluster update enables Design to provide high quality of clustering with very few iterations. Our evaluation shows that Design provides comparable accuracy to state-of-the-art clustering algorithms, while providing 4.7× faster and 5.8× higher energy efficiency.**

I. Introduction

With the emergence of the Internet of Things (IoT), sensory and embedded devices generate massive data streams and demand services that pose huge technical challenges due to limited device resources. Today IoT applications analyze raw data by running machine learning algorithms. Since the majority of data generated are not associated with any labels, clustering algorithms are the most popular learning methods used for data analysis [1]. Clustering algorithms are unsupervised and have applications in many fields including machine learning, pattern recognition, image analysis, information retrieval, bioinformatics, data compression, and computer graphics [2], [3], [4], [5]. These algorithms are used to group a set of objects into different classes, so that objects within the same class are similar to each other.

The process of clustering datasets involves heavy computations as most algorithms need to calculate pairwise distances between all the points in the dataset [6], [7], [8]. To give non-linearity to clustering task, most existing algorithms are using complex distance metric, e.g., Euclidean distance, and use iterative process for clustering. However, computing these distance metrics is extremely expensive on today's computers. In addition, the clustering requires many iterations in order to perform clustering.

To achieve real-time performance with high energy efficiency and robustness, we rethink not only how we accelerate clustering algorithms in hardware, but also to redesign the algorithms using strategies that more closely model *the human brain*. Hyperdimensional (HD) computing [9] is based on a short-term human memory model, Sparse distributed memory, emerged from theoretical neuroscience [10]. HD computing is motivated by a biological observation that the human brain operates on a *robust high-dimensional* representation of data originated from the large size of brain circuits [9]. It thereby models human memory using points of a high-dimensional space. HD computing mimics several important functionalities of the human memory model with vector operations which are computationally tractable and mathematically rigorous in describing human cognition. Many studies show the potential of HD computing for diverse applications, e.g., classification [11], bioinformatics [12], [13], and robotics [14]. The research work in [15] tried to use HD computing for clustering. However, the lack of proper encoding and learning process results in low quality of clustering even using complex and costly cosine similarity metric.

In this paper, we proposed Design, a novel framework for efficient and fully binary clustering in high-dimensional space. To the best of our knowledge, Design is the first fully binary HD-based clustering approach that enables very high quality of clustering. The main contribution of this paper is listed in the following:

- Design introduces a non-linear encoding approach which maps data points into high-dimensional space. Thanks to non-linearity of this encoder, the clustering task can perform using simple and hardware-friendly metrics, e.g., Hamming distance.
- Design propose a new clustering framework enabling the majority of HD computing to perform on binary space. Design binarizes encode data and the cluster centers, enabling the similarity search to perform using simple and efficient Hamming distance metric. To give a higher flexibility to model to training, Design stores a non-bianrized cluster centers and use that model for center update.
- Thanks to HD transparent model, Design provides a confident on prediction about the correct cluster of each input data. During the iterative clustering process, Design updates the centers adaptively, based on the confidence of each data point on predicting the correct center. This results in significant reduction in the number of required iterations to learn.
- We evaluate Design efficiency on a wide range of clustering datasets. As compared to the state-of-the-art clustering algorithm, Design provides comparable quality of clustering while providing 4.7× faster and 5.8× higher energy efficiency.

II. Related Work

Prior research applied HD computing into diverse cognitive tasks such as classification [16], robotics [14], and bioinformatics [17]. Most recent focus of the HD computing approach is on classification task. Several recent work proposed optimizations for binarzing, quantizing, and sparsifying of the HD classification model. The main goal of these approaches is to simplify the similarity search to hardware-friendly distance metrics. Prior work also focused on designing accelerator for HD classification. Several existing works have presented the hardware for efficiently processing HD classification, including both FPGA and in-memory accelerators [18], [19], [20]. For example, work in [21] implemented HD computing on FPGA, but it only works with binary hypervectors with applications limited to the classification of text and time-series signals.

Although there are several frameworks for efficient classification [22], [23], there is no suitable framework for unsupervised learning. There are major issues with existing clustering algorithms. clustering requires to map data points into hypervectors with integer representation in order to provide high accuracy. Although mapping only happens once, the rest of clustering steps are

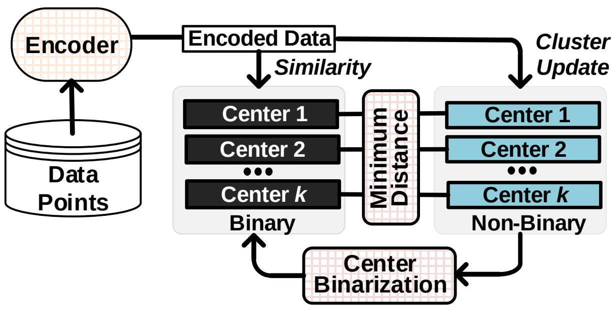

Fig. 1. Overview of Design clustering framework.

Algorithm 1 Design Encoding

```
1:  function ENCODER($\vec{F}$,$\vec{H^b}$)
2:      $\vec{B}_1,\dots,\vec{B}_D \leftarrow Normal(\mu=0,\sigma=1,\dim=n)$
3:      $\vec{b}=(b_1,\dots,b_D) \leftarrow Uniform(a=0,b=2\pi,\dim=D)$
4:      for $i=1$ to $n$ do
5:          for $j=1$ to $D$ do
6:              $h_{ij} \leftarrow \cos(\vec{B}_j\cdot\vec{F}_i+b_j)\times\sin(\vec{B}_j\cdot\vec{F}_i)$
7:          end for
8:          $\vec{H}_i \leftarrow (h_{i1},\dots,h_{iD})$            ▷ Encoded hypervector
9:          $\vec{H}_i^b \leftarrow sign(\vec{H})$            ▷ Encoded binary hypervector
10:     end for
```

significantly costly over integer vectors. In addition, the similarity search, which is the most common operation of clustering, needs to happen using cosine metric. For example, work in [15] used HD computing for HD-based clustering. However, the lack of proper encoding and learning significantly reduces the quality of clustering, even using costly cosine metric. To the best of our knowledge, we proposed Design, the first framework for fully binary clustering in high-dimensional space. Design introduces a new non-linear encoding and an adaptive clustering approach that simplifies the entire computation to binary similarity search.

III. Design: Clustering in Binary Space

Figure 1 shows the overview of Design framework. Design first maps data points into high-dimensional space. This encoding module needs to consider the relation between different features and map data point into non-linear space where the similarity could preserve using Hamming metric. The Design performs clustering over the encoded data. Design generates k binary random hypervector representing k cluster centers. Design compares the Hamming distance similarity of each encoded data point with the cluster center. Each data point gets the label of a center that it has highest similarity with it. By going through the entire dataset, or a batch, Design assigns a label to each data point. Then, Design updates the cluster centers depending on Design confidence for each clustering task. The cluster update makes the centers to get non-binary elements. This eleminates the Hamming distance similary search during the next iterations. To address this, Design binarizes the new centers, while still stores non-binary centers. The new binary centers are used for the similarity search and assigning a label to each data points. Depending on those labeled, Design update non-binary model. This iterative training continues until the the number of labels do not change during few iterations.

A. Non-Linear Encoding

There are multiple encoding methods proposed in literature [24], [25], [26]. Although these methods have shown excellent classification accuracy for their application-specific problems, to the best of our knowledge, the existing encoding methods linearly combine the hypervectors corresponding to each feature, resulting in suboptimal classification quality for general classification problems. To obtain the most informative hypervectors, the HD encoding should consider the non-linear interactions between the feature values with different weights.

In this context, we propose a novel encoding method which exploits the kernel trick [27], [28] to map data points into the high-dimensional space. The underlying idea of the kernel trick is that data, which is not linearly separable in original dimensions, might be linearly separable in higher dimensions. Let us consider certain functions $K(x,y)$ which are equivalent to the dot product in a different space, such that $K(x,y)=\Phi(x)\cdot\Phi(y)$, where $\Phi(\cdot)$ is often a function for high dimensional projection. The Radial Basis Function or Gaussian Kernel is the most popular kernel:

$$K(x,y)=e^{\frac{-||x-y||^2}{2\sigma^2}}$$

We can take advantage of this implicit mapping by replacing their decision function with a weighted sum of kernels:

$$f(\cdot)=\sum_{i=0}^{N} c_i K(\cdot,x_i)$$

where (x_i,y_i) is the training data sample, and the c_is are constant weights. The study in [27] showed that the inner product can efficiently approximate Radial Basis Function (RBF) kernel, such that:

$$K(x,y)=\Phi(x)\cdot\Phi(y)\approx z(x)\cdot z(y)$$

The Gaussian kernel function can now be approximated by the dot product of two vectors, $z(x)$ and $z(y)$.

The proposed encoding method is inspired by the RBF kernel trick method. Algorithm 1 shows our encoding procedure. Let us consider an encoding function that maps a feature vector $\vec{F}=\{f_1,\ f_2,\dots,\ f_n\}$, with n features ($f_i\in\mathbb{N}$) to a hypervector $\vec{H}^b=\{h_1,\ h_2,\dots,\ h_D\}$ with D dimensions ($h_i^b\in\{-1,1\}$). We generate each dimension of the encoded data by calculating a dot product of the feature vector with a randomly generated vector as $h_i=\cos(\vec{B}_i\cdot\vec{F}+b_j)\times\sin(\vec{B}_i\cdot\vec{F})$, where B_i is the randomly generated vector with a Gaussian distribution (mean $\mu=0$ and standard deviation $\sigma=1$) with the same dimensionality to that of the feature vector. The random vectors $\{\vec{B}_1,\vec{B}_2,\cdots,\vec{B}_D\}$ can be generated once offline and then can be used for the rest of the clustering task. After this step, each element h_i of a hypervector $\vec{H}$ has a non-binary value. In the HD computing, binary (bipolar) hypervectors are often used for the computation efficiency. We thus obtain the final encoded hypervector by binarizing it with a sign function ($\vec{H}^b=sign(\vec{H})$) where the sign function assigns all positive hypervector dimensions to '1' and zero/negative dimensions to '-1'. The encoded hypervector stores the information of each original data point with D bits.

B. Initial Center Generation

Design starts clustering from initial centers. Due to the randomness and sparsity in high-dimensional space, Design starts clustering by generating k random cluster centers $\{\vec{C}_1,\vec{C}_2,\cdots,\vec{C}_k\}$. Since these vectors are randomly generation, they are nearly orthogonal, meaning that:

$$\delta(\vec{C}_i,\vec{C}_j)\approx 0 \quad 1\le i,j\le k \ \ \&\ \ i\neq j$$

where δ is a cosine similarity metric. The random selection of centers ensures that the centers are not biased toward any specific center/data point.

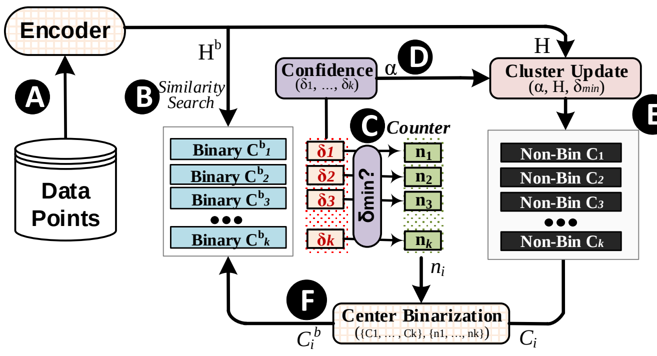

Fig. 2. Design clustering framework.

C. Similarity Search

The next step of Design clustering is to assign each encoded data point to one of the cluster centers (B). This is equivalent to unsupervised labeling of encoded data based on their similarity to cluster centers. Depending on the goal of clustering and representation of centers/data point, clustering algorithms are using different metrics for similarity search. For example, K-means clustering usually uses costly Euclidean distance as similarity metric [29], [30]. Thanks to Design encoding (explained in Section III-A), the similarity metric can be simplifies to cosine or Hamming distance. Prior work showed that in order to get high quality of clustering, the values and centers need to represent using non-binary (e.g., Integer or floating-point) values, resulting in using expensive cosine metric for similarity search. The main goal of our proposed Design is to simplify the values representation to binary and similarity metric to Hamming distance, with no or minor impact on the clustering quality.

As Algorithm 1 described, Design encoding module generates both non-binary ($\vec{H}$) and binary ($\vec{H}^b$) data. Similarly, Design stores two copy of the cluster centers: (i) an original version of centers with non-binary representations ($\vec{C}_i$), and (ii) a binarized version of the cluster centers ($\vec{C}_i^b$). Note that in the first iteration both cluster centers are the same, since we initialized centers are binary hypervectors (shown in Figure 2)

During the clustering, Design checks the Hamming similarity of the binary data with the binary cluster center ($\delta_i = \delta(\vec{H}^b, \vec{C}_i^b)$). The output of this search is a center that has the highest similarity with a data point (C):

$$C_{max} \leftarrow MIN_{i \in K}(\delta_i)$$

The same search assigns all data points to one of the centers depending on their Hamming similarity. Thanks to Design transparent model, it also provides a confidence level for each data point during the similarity search. This confidence rate specifies how clear a data point is assigned to a center (D). In Section III-D, we show how this factor can be used to enhance and speedup clustering process. For each data point, confidence is defined as:

$$\alpha = \frac{\delta_{max} - mean(\delta_i)}{std(\delta_i)} \quad 1 \le i \le k$$

Where δ_{max} is a cluster with highest similarity with data, $std(\delta_i)$ shows the standard division of the binary encoded data ($\vec{H}^b$) with other centers, and $mean(\delta_i)$ shows the average Hamming distance of a binary encoded data with all centers.

D. Center Update

In the first iteration, Design provides a low quality of clustering as the initial cluster centers are often not a good representation of data. To enhance the quality of clustering, we update the centers based on the new data point assigned to each center. Existing clustering approaches, e.g., K-means, compute the new centers by averaging over all training data point corresponding to each cluster center. However, all data points assigned to a cluster do not have the same confidence about the correctness of their label (assigned center). This naive center update increases the number of clustering iterations, as we require many iterations to learn a specific pattern. In contrast, Design propose a method to find the new cluster centers based on a confidence that each data point had during the similarity search to select a center.

Each cluster center updates based on all data points assigned to each center as well as the confidence level of each data point. Design has multiple options for cluster update by accumulating the binary or non-binary query data based on their corresponding confidence level or in non-adaptive way (E). The metrics are given here:

- *Baseline Non-Adaptive Update*: $\vec{C}_i = \sum \vec{H}_i$
- *Adaptive Non-binary Update*: $\vec{C}_i = \sum \alpha \vec{H}_i$
- *Adaptive Binary Update*: $\vec{C}_i = \sum \alpha \vec{H}_i^b$

Where H_i and H_i^b are all non-binary and binary query data that assign to cluster i^{th} center during the similarity search. The same update happens for all cluster centers. Note that the update always happen over the non-binary centers.

Next, Design needs to update the binary centers based on the newly updated non-binary model. The goal of this update is to generate a new binary centers that can better represent the cluster centers based on the distribution of the data. Design updates the binary cluster by simply binarizing every elements of the non-binary centers. when using bipolar representation (hypervectors in $\{-1,+1\}^D$), this binarization is a simple `sign()` function that assigns negative and positive elements to 0 and 1, respectively.

In Design with binary representation, which is desired for hardware, we apply a *majority* function on non-binarized class hypervectors (F). Given a center hypervector, $\vec{C} = \langle c_D, \cdots, c_1 \rangle$, the majority function is defined as follows:

$$MAJ(\vec{C}_i,\, n_i) = \langle c_D^b, \cdots, c_1^b \rangle \text{ where } c_j^b = \begin{cases} 0, & \text{if } c_j < n_i/2 \\ 1, & \text{otherwise.} \end{cases}$$

Using the majority function, the final hypervector for each data point is encoded by $\vec{C}^b = MAJ(\vec{C}_i, n_i/2)$, and $\vec{C}^b \in \{0,1\}^D$, and n_i is a counter corresponding to each center to keep tracks of the number of data points assign to the i^{th} center.

E. Iterative Learning

Design performs an iterative process for clustering. In each iteration, Design computes the similarity of a binary data with the binary centers and then updates the non-binary centers depending the search results and the search confidence. Then, we binarize the updated centers in order to find a new binary centers. After every iterations, Design checks the changes on the cluster centers in order to decide the convergences. Design is converged when all binary cluster centers have less than 2% changes in their elements during two consecutive iterations. Note that each Design iteration can happen over the entire dataset or a batch of data. Using a large batch can potentially provide higher quality of clustering, as the centers can be updated after looking at the entire data points. In contrast, a small batch size speeds up Design computation by reducing the number of iterations, although it may not provide the highest possible quality of clustering. In section V-F, we explore

Algorithm 2 Design clustering process

```
1:  function DESIGN($\vec{H}, \vec{H}^b, k$)
2:      $\vec{C}_i \leftarrow Random\{\vec{H}_1, \ldots, \vec{H}_n\}$        ▷ Compute initial centers
3:      $\vec{C}_i^b \leftarrow sign(\vec{C}_i)$                               ▷ Binary centers
4:      for $it = 1$ to maxiter do
5:          for $i = 1$ to $n$ do
6:              $\delta_j \leftarrow \delta(\vec{C}_j^b, \vec{H}_i^b)$          ▷ For $1 \le j \le k$
7:              $\delta_{\max} \leftarrow \min_{j \in K} \delta_j$              ▷ Where $K = \{1, \ldots, k\}$
8:              $\alpha \leftarrow (\delta_{\max} - mean(\delta_j))/std(\delta_j)$
9:              $\vec{C}_{\delta_{\max}} \leftarrow \vec{C}_{\delta_{\max}} + \alpha \vec{H}_i$   ▷ Update model
10:             $\vec{C}_{\delta_{\max}}^b \leftarrow sign(\vec{C}_{\delta_{\max}})$
11:         end for                                                         ▷ Converge check
12:         if No Change on Centers? then return
13:         end if
14:     end for
```

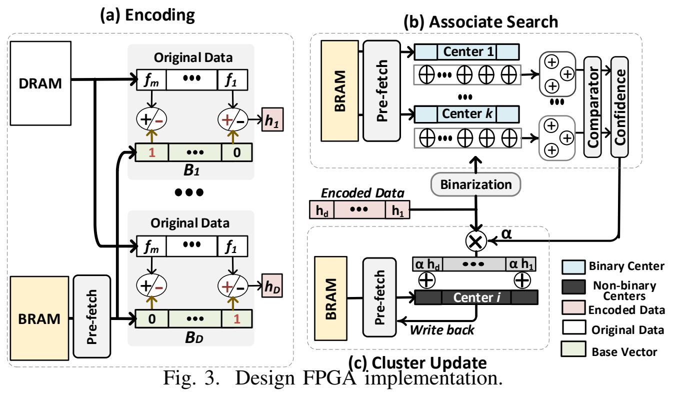

Fig. 3. Design FPGA implementation.

the impact of batch size on Design efficiency and quality of clustering.

The algorithm 2 describes Design clustering process using adaptive non-binary cluster update. Our algorithm takes binary ($\vec{H}^b$) and non-binary ($\vec{H}$) encoded hypervectors as inputs and return assigns each input to one of the k cluster centers.

IV. Hardware Acceleration

Design can be implemented in different platforms such as CPU, GPU, or FPGA. Due to a large amount of bit-wise operations exist in training and inference of HD computation, FPGA is a suitable candidate for efficient HD computing acceleration. Design has three main phases: encoding, associative search, and cluster update. Here, we briefly discuss about the implementation of each block on FPGA. Figure 3 shows details of our FPGA implementation.

A. Encoding

The encoding happens by multiplying the input data with a pre-generated projection matrix. To provide high computation efficiency, we use projection matrix with binary elements. This simplifies the encoding task to addition and subtraction of different input features. As Figure 3a shows, our implementation reads m features of original data from DRAM. Due to limited bandwidth of off-chip DRAM, the m is usually a portion of the total number of input features. Similarly, we pre-stores all binary base hypervectors $\{B_1, \cdots, B_D\}$ in the BRAM block. Depending on the value of each base hypervector, our design adds or subtracts the m features. The encoding computation is performed using FPGA Look-Up Tables (LUTs) resources. In the next cycle, FPGA reads the next m features and encode them based on the base values in those new

TABLE I
Datasets

	Data	*Features*	*Clusters*	*Description*
DIM	1024	128	16	Gaussian Data in high-dimension [33]
A3	7500	2	50	Low-dimensional distributed data [34]
TETRA	236	16	9	Genetic tetragonula bees [35]
MNIST	70000	784	10	Handwritten digit recognition [36]
VERONICA	207	586	2	Genetic AFLP of Veronica plants [37]
UCIHAR	10299	561	6	Human Activity Recognition [38]
SYNTHET I	1000	100	25	Synthetic Data
SYNTHET II	100000	100	25	Synthetic Data

dimensions. Depending on the number of features or dimensionality of the encode data, the throughput of our implementation can be bounded by DRAM bandwidth or FPGA LUT resources.

B. Associative Search

Our implementation performs the similarity search using Hamming distance metric. As shown in Figure 3b, this functionality can be implemented using an XOR array that computes the difference of encoded data with binary cluster centres. Our implementation accumulates the XOR results over D dimensions in order to find a Hamming distance of a data with each center. Finally, we compute the confidence level depending on all k Hamming distance values, each correspond to a cluster center. Thanks to Design similarity metric, the associative search is implemented using FPGA LUTs in a highly parallel and efficient way.

C. Cluster update

For cluster update, Design keeps two copies of the center in BRAM blocks: a non-binary and a binary. In contrast to associative search that uses binary centers, the cluster update performs computation over non-binary centers. As Figure 3c shows, our design updates the non-binary center by adding the weighted encoded data ($\alpha \times \vec{H}$) to a center with the highest Hamming similarity. After each cluster update, the non-binary cluster is written back to FPGA BRAMs.

Note that our implementation works in a pipeline in order to maximize the resource utilization. When FPGA encodes i^{th} data point, the associative search computes the similarity of the $i-1^{th}$ data with the centers. At the same time, FPGA updates one of the non-binary centers based on the similarity search of the $i-2^{th}$ data.

V. Evaluation

A. Experimental Setup

We used C++ software implementation for Design learning and verification. We exploit Scikit-learn library [31] for implementing other clustering approaches. For hardware support, we fully implement Design functionality in RTL using Verilog HDL. Design is deeply pipelined to run with 200 MHz clock frequency. We verify the timing and functionality of the Design using both synthesis and real implementation on Xilinx Vivado Design Suite [32]. To estimate the power consumption of the FPGA we use the builtin Xilinx Power Estimation tool in Vivado Design suite. Design is implemented on the Kintex-7 FPGA KC705 Evaluation Kit. We examine Design efficiency and quality of clustering on several large-scale datasets including actual and synthetic datasets. Table I lists all popular datasets. We also selected two synthetic random data with 25 cluster centers, radius range of $[r_l, r_h] = [0..\sqrt{2}, \sqrt{2}..\sqrt{32}]$, and noise rate of 0-10%. To measure cluster quality, we rely on correct labels of data points and find out how many points were classified in a cluster that does not reflect the label associated with the point.

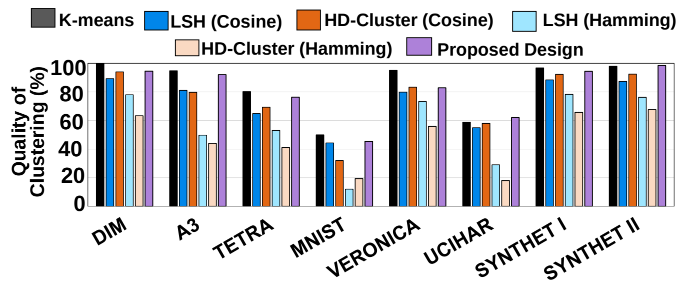

Fig. 4. Comparison of Design quality of clustering with other algorithms.

B. *Design vs Other Clustering Algorithms*

Figure 4a compares Design quality of clustering with k-means and the state-of-the-art clustering approaches in high-dimensional space: HD-based clustering (HD-cluster) [15], and clustering based on Locality Sensitive Hashing (LSH-cluster) [39], [40]. K-means algorithm is working on original data and use Euclidean distance as a similarity metric. Other approaches map data points into $D = 4k$ dimensions before performing clustering. For LSH and HD-based clustering, the results are reported using both cosine and Hamming distance metrics. These approaches are using integer values and cosine metric, while Hamming distance similarity metric is used when we use binary vectors. Our evaluation shows that Design provides comparable quality of clustering to k-means.In addition, as compared to HD-cluster and LSH-cluster, Design provides significantly higher quality of the clustering. Comparing with HD and LSH-based approaches with Hamming distance similarity metric, which have similar computation complexity as Design, Design provides 24.3% and 33.5% higher quality of cluster, respectively. Even using cosine metric, HD-cluster and LSH-cluster provide 7.5% and 8.9% lower quality of clustering as compared to Design. This improvement comes from: (i) a non-linear encoder that better preserves the similarity of values in high-dimensional space using hardware-friendly Hamming distance, and (ii) adaptive iterative learning to update the cluster centers based on the confidence of each data point.

C. *Design Efficiency*

Figure 5 shows the energy consumption and execution time of Design, LSH, and HD-cluster running on GTX 1080 GPU. All results are normalized to k-means energy and execution time. For LSH and HD-cluster, the results are reported when they use cosine (LSH-cos and HD-cos) and Hamming distance (LSH-Ham and HD-Ham) as similarity metric. Our evaluation shows that Design provides significantly higher efficiency as compared to k-means and other clustering approaches. For example, Design provides on average 5.8× higher energy efficiency and 4.7× speedup as compared to k-means. Similarly, Design provides significantly higher efficiency as compared to LSH-cos and HD-cos. This higher efficiency comes from: (i) Design capability to perform the majority of clustering operations using efficient binary operations. For example, Design support similarity search, which is the most frequent clustering functionality, using hardware-friendly Hamming distance metric. (ii) Design significantly reduces the number of retraining iterations. In contrast to K-means that train a model in hundreds of iterations, Design get similar quality of clustering on average in 5× lower number of iterations.

Design also provides higher efficiency than both LSH-Hamming and HD-Hamming. This efficiency does not come from from simplicity of Design clustering. Design uses more complex encoding than both LSH and HD-based clustering approaches. As Figure 5 shows, Design encoding takes 36% of total energy, while this portion is less than 20% and 11% in LSH-Hamming and

TABLE II
DESIGN EFFICIENCY AND QUALITY IN DIFFERENT CONFIGURATIONS

		DIM	*A3*	*TETRA*	*MNIST*	*VERONICA*	*UCIHAR*
Non-adaptive	*Quality*	94.7%	88.2%	69.9%	44.9%	83.8%	57.7%
	Iterations	23.9	82.7	28.3	92.1	15.6	39.3
Adaptive non-binary	*Quality*	95.9%	92.3%	73.1%	46.2%	88.0%	62.1%
	Iterations	6.2	17.1	7.4	16.3	4.2	15.2
Adaptive Binary	*Quality*	94.0%	91.0%	72.7%	47.1%	86.0%	61.7%
	Iterations	6.9	26.3	9.2	19.3	6.8	21.1

HD-Hamming. Therefore, in terms of a single iteration, Design consumes about 1.2× and 1.3× higher energy than LSH-Hamming and HD-Hamming. However, Design significantly reduces the number of required clustering iterations. As we explained in Section III-D, Design adaptive clustering update reduces the number of required iterations. This number has a direct impact on speeding up and improving clustering efficiency. Our evaluation shows that Design provides on average 1.6× and 1.4× higher energy efficiency as compared to LSH-Hamming and HD-Hamming, respectively. We provide more detailed discussion about adaptive cluster update on Section V-D.

Figure 6 compares Design computation efficiency with K-means, LSH, and HD-cluster running on GTX 1080 GPU and Xilinx Kintex-7 FPGA. For K-means, we used the implementation of work in [] which fully utilizes FPGA to maximize the throughput. For fairness, we report HD and LSH-based clustering only for cosine metric (LSH-cos, anmd HD-cos), as this configuration provides comparable accuracy to Design. Our evaluation shows that algorithm using non-binary representation have minor efficiency improvement on FPGA, especially in performance. For k-means, HD-cosine, and LSH-cosine, the similarity search is performed using complex metrics. These metrics need to utilize the limited FPGA DSPs for hardware implementation, resulting in low computation efficiency. In contrast, Design can preform majority of computation using bitwise operations, e.g., Hamming distance similarity search, that can effectively parallelize on the FPGA Look-Up Tables (LUTs). Our evaluation shows that Design on FPGA can provide 6.2× and 91.× (6.2× and 7.5×) faster and higher energy efficiency as compared to K-means (HD-cosine) running on FPGA.

D. *Design & adaptive cluster Update*

We compare Design quality of clustering using different cluster update methods introduced in Section III-D: (i) non-adaptive update, (ii) adaptive non-binary update, and (iii) adaptive binary update. Table II compares Design in these configurations from quality of clustering and the number of required iterations to converge. Our evaluation shows that Design with non-adaptive update requires the maximum number of iterations and also provides lowest quality of clustering. This approach does not consider the confidence factor (explained in Section sec:) to update the model, thus it gives the same chance to all data points regardless of how well/confident they assign to a center. Design using adaptive update provides higher quality of clustering and faster convergence. In the same dimensionality, the non-binary update provides higher quality of clustering and converges with lower number of iterations. However, the binary update has much lower impact on updating the cluster centers, results in reducing the number of iterations and possibly degrading the quality of clustering. In terms of efficiency, adaptive update provides significantly higher clustering efficiency than non-adaptive approach. Comparing non-binary and binary adaptive update, Design with binary update provides higher efficiency in a single learning iteration. However, the higher required iterations of binary update reduces the overall efficiency as compared to Design using non-binary update.

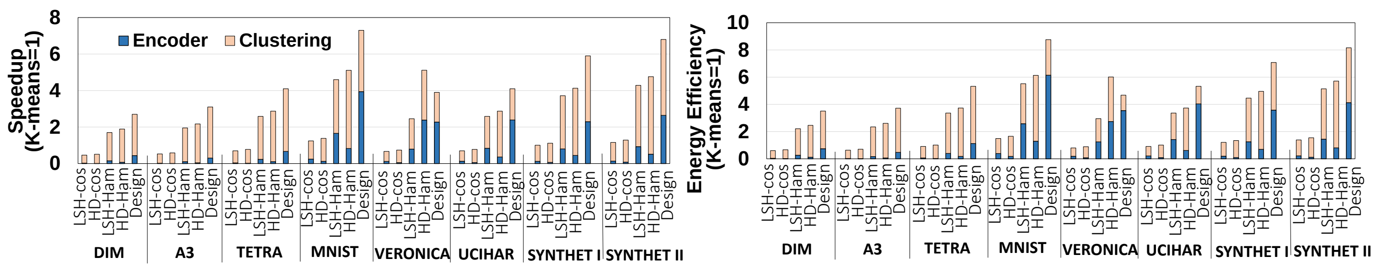

Fig. 5. Design computation efficiency as compared to other clustering algorithms.

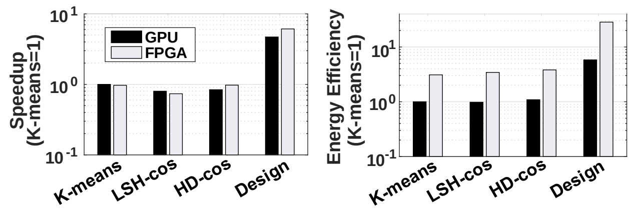

Fig. 6. Design efficiency on GPU vs FPGA.

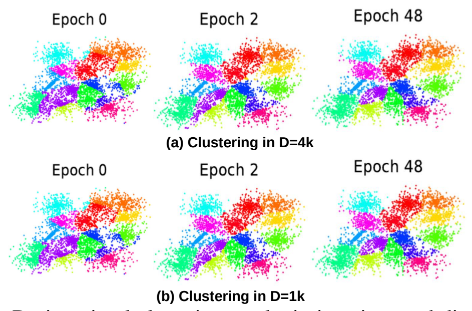

Fig. 7. Design visual clustering results in iterations and dimensions.

E. Design & Dimensionality

Table III shows the impact of dimension reduction of Design efficiency and quality of clustering. The results are reported for Design with adaptive non-binary update. Our evaluation shows that Design has high robustness to reducing the hypervector dimensionality. For example, for most tested datasets, Design provide maximum accuracy using $D = 3k$. However, further reducing dimensionality results in quality loss. For example, using $D = 1k$ ($D = 0.5k$), Design provides on average 2.9% (5.4%) quality loss as compared to using fully dimensionality ($D = 4k$). In terms of efficiency, reducing dimensionality has direct impact on Design computation efficiency. The efficiency improvement caused by hypervector dimensionality is often over linear, as high dimensional vectors suffer from limited on-chip memory. This results in creating stall cycles on FPGA computation. From other hand, lower dimensional vectors require more iterations to converge. As Table III reports, the number of required iterations increases using vectors with lower dimensionality. For example, using vectors with $D = 4k$, Design converges on average in 12.8 iterations, while this number increases to 23.1 (28.7) iterations using $D = 1k$ ($D = 0.5k$). The increase in learning iterations further improve Design computation cost when using low dimensionality. We observe that Design using $D = 3k$ balances the efficiency of each iteration as well as the number of iterations, resulting in maximum efficiency.

Figure 7a visualizes Design quality of clustering during different learning iterations. The results are reported for Design using two different dimensions. As the figure shows, Design quality of clustering improves during different iterations and also by increasing dimensionality.

TABLE III
IMPACT OF DIMENSIONALITY ON DESIGN NUMBER OF ITERATIONS AND QUALITY OF CLUSTERING.

Dimensions (D)		*4k*	*3k*	*2k*	*1k*	*0.5k*
DIM	*Quality*	95.2%	95.1%	94.7%	93.8%	92.3%
	Iterations	6.9	7.8	8.2	11.4	13.2
A3	*Quality*	91.0%	90.9%	90.5%	88.8%	86.1%
	Iterations	26.3	31.4	44.2	47.3	58.4
TETRA	*Quality*	72.7%	72.5%	71.3%	70.2%	67.2%
	Iterations	9.2	9.3	9.7	10.2	10.7
MNIST	*Quality*	47.1%	46.8%	44.4%	38.3%	34.7%
	Iterations	19.3	21.5	33.8	46.6	63.9
VERONICA	*Quality*	87.6%	87.2%	85.3%	84.9%	82.6%
	Iterations	6.8	7.5	8.0	8.8	9.2
UCIHAR	*Quality*	61.7%	61.2%	60.9%	58.0%	56.8%
	Iterations	21.1	24.5	32.7	37.2	45.7

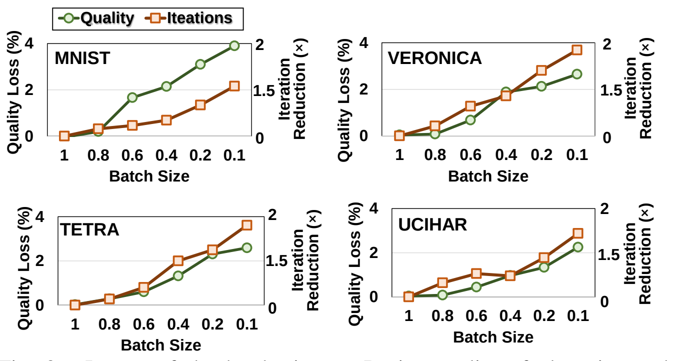

Fig. 8. Impact of the batch size on Design quality of clustering and convergence.

F. Design Batch Size Trade-offs

To speedup Design clustering speed, one can decide to update the cluster centers more frequently without cover all data points in each iteration. Figure 8 shows the impact of batch size on Design efficiency and quality of clustering. The results are shown for Design using batch size equivalent to complete or a portion of dataset. For example, $B = 0.4$ means that the batch size is half of the number of data points. The main motivation of reducing the batch size is to quickly update the model before waiting to see the entire data points. This approach enables us to locally update the center for each batch. A small batch size does not provide the same chance for all data points to vote and update the model. This approach speeds up Design computing by reducing the number of iterations. However, it can potentially result in lower quality of clustering. Our evaluation shows that Design with $B = 0.6$ ($D = 0.2$) achieves $1.2\times$ ($1.6\times$) less number of iterations to converge while providing 0.8% (2.2%) quality loss.

VI. Conclusion

In this paper, we proposed Design, a novel adaptive clustering in high-dimensional space. maps data points into high-dimensional space and perform adaptive iterative learning to perform clustering. Design performs clustering by assigning each data point to a set of initialzed centers. Then, it updates the centers adaptively based on: (i) data points assigned to each cluster, and (ii) how confident they have been match with each center. This adaptive cluster update enables Design to provied high quality of clustering with very few iterations. In addition, Design simplifies the similarity search, the most costly/frequent clustering operation, to Hamming distance which can be accelerated in today's hardware. Our evaluation shows that Design provides comparable accuracy to state-of-the-art clustering algorithms, while providing 4.7× faster and 5.8× higher energy efficiency.

References

[1] D. Singh and C. K. Reddy, "A survey on platforms for big data analytics," *Journal of big data*, vol. 2, no. 1, p. 8, 2015.

[2] A. K. Jain, "Data clustering: 50 years beyond k-means," *Pattern recognition letters*, vol. 31, no. 8, pp. 651–666, 2010.

[3] J. A. Silva, E. R. Faria, R. C. Barros, E. R. Hruschka, A. C. De Carvalho, and J. Gama, "Data stream clustering: A survey," *ACM Computing Surveys (CSUR)*, vol. 46, no. 1, p. 13, 2013.

[4] E. P. Xing, M. I. Jordan, S. J. Russell, and A. Y. Ng, "Distance metric learning with application to clustering with side-information," in *Advances in neural information processing systems*, pp. 521–528, 2003.

[5] C. C. Aggarwal and C. Zhai, *Mining text data*. Springer Science & Business Media, 2012.

[6] L. Fu, B. Niu, Z. Zhu, S. Wu, and W. Li, "Cd-hit: accelerated for clustering the next-generation sequencing data," *Bioinformatics*, vol. 28, no. 23, pp. 3150–3152, 2012.

[7] X. Cai, F. Nie, and H. Huang, "Multi-view k-means clustering on big data," in *Twenty-Third International Joint conference on artificial intelligence*, 2013.

[8] A. Fahad, N. Alshatri, Z. Tari, A. Alamri, I. Khalil, A. Y. Zomaya, S. Foufou, and A. Bouras, "A survey of clustering algorithms for big data: Taxonomy and empirical analysis," *IEEE transactions on emerging topics in computing*, vol. 2, no. 3, pp. 267–279, 2014.

[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] P. Kanerva, *Sparse distributed memory*. MIT press, 1988.

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

[12] 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.

[13] 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.

[14] A. Mitrokhin, P. Sutor, C. Fermüller, and Y. Aloimonos, "Learning sensorimotor control with neuromorphic sensors: Toward hyperdimensional active perception," *Science Robotics*, vol. 4, no. 30, p. eaaw6736, 2019.

[15] M. Imani, Y. Kim, T. Worley, S. Gupta, and T. Rosing, "Hdcluster: An accurate clustering using brain-inspired high-dimensional computing," in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1591–1594, IEEE, 2019.

[16] 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, 2016.

[17] Y. Kim, M. Imani, N. Moshiri, and T. Rosing, "Geniehd: Efficient dna pattern matching accelerator using hyperdimensional computing," in *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, IEEE, 2020.

[18] T. F. Wu *et al.*, "Brain-inspired computing exploiting carbon nanotube fets and resistive ram: Hyperdimensional computing case study," in *2018 IEEE International Solid-State Circuits Conference-(ISSCC)*, pp. 492–494, IEEE, 2018.

[21] 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.

[19] 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.

[20] S. Salamat, M. Imani, B. Khaleghi, and T. Rosing, "F5-hd: Fast flexible fpga-based framework for refreshing hyperdimensional computing," in *FPGA*, pp. 53–62, ACM, 2019.

[22] 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.

[23] M. Imani, S. Salamat, B. Khaleghi, M. Samragh, F. Koushanfar, and T. Rosing, "Sparsehd: Algorithm-hardware co-optimization for efficient high-dimensional computing," in *2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pp. 190–198, IEEE, 2019.

[24] 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.

[25] 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.

[26] 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.

[27] A. Rahimi and B. Recht, "Random features for large-scale kernel machines," in *Advances in neural information processing systems*, pp. 1177–1184, 2008.

[28] B. Schölkopf, "The kernel trick for distances," in *Advances in neural information processing systems*, pp. 301–307, 2001.

[29] N. Dhanachandra, K. Manglem, and Y. J. Chanu, "Image segmentation using k-means clustering algorithm and subtractive clustering algorithm," *Procedia Computer Science*, vol. 54, pp. 764–771, 2015.

[30] H. Koga, T. Ishibashi, and T. Watanabe, "Fast agglomerative hierarchical clustering algorithm using locality-sensitive hashing," *Knowledge and Information Systems*, vol. 12, no. 1, pp. 25–53, 2007.

[31] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, *et al.*, "Scikit-learn: Machine learning in python," *Journal of Machine Learning Research*, vol. 12, no. Oct, pp. 2825–2830, 2011.

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

[33] P. Franti, O. Virmajoki, and V. Hautamaki, "Fast agglomerative clustering using a k-nearest neighbor graph," *IEEE transactions on pattern analysis and machine intelligence*, vol. 28, no. 11, pp. 1875–1881, 2006.

[34] I. Kärkkäinen and P. Fränti, *Dynamic local search algorithm for the clustering problem.* University of Joensuu Joensuu, Finland, 2002.

[35] P. Franck, E. Cameron, G. Good, J.-Y. Rasplus, and B. Oldroyd, "Nest architecture and genetic differentiation in a species complex of australian stingless bees," *Molecular Ecology*, vol. 13, no. 8, pp. 2317–2331, 2004.

[36] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[37] M. M. Martínez-Ortega, L. Delgado, D. C. Albach, J. A. Elena-Rosselló, and E. Rico, "Species boundaries and phylogeographic patterns in cryptic taxa inferred from aflp markers: Veronica subgen. pentasepalae (scrophulariaceae) in the western mediterranean," *Systematic Botany*, vol. 29, no. 4, pp. 965–986, 2004.

[38] 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.

[39] "Lsh clustering." https://github.com/usc-isi-i2/dig-lsh-clustering.

[40] X. Shen, W. Liu, I. Tsang, F. Shen, and Q.-S. Sun, "Compressed k-means for large-scale clustering," in *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.