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

Alejandro Hernández-Cano
School of Science
Universidad Nacional Autónoma de México
ale.hdz333@ciencias.unam.mx

Yeseong Kim
Daegu Gyeongbuk Institute of Science and Technology
yeseongkim@dgist.ac.kr

Mohsen Imani
Department of Computer Science
University of California, Irvine
m.imani@uci.edu

***Abstract*—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, especially in healthcare. However, running clustering algorithms on embedded devices is significantly slow as the computation involves a large amount of complex pairwise similarity measurements. In this paper, we proposed FebHD, an adaptive framework for efficient and fully binary clustering in high-dimensional space. Instead of using complex similarity metrics, e.g., Euclidean distance, FebHD introduces a non-linear encoder to map data points into sparse high-dimensional space. FebHD encoder simplifies the similarity search, the most costly and frequent clustering operation, to Hamming distance, which can be accelerated in today's hardware. FebHD performs clustering by assigning each data point to a set of initialized centers. It then updates the centers adaptively based on: (i) data points assigned to each cluster, and (ii) the confidence of the model on the clustering prediction. This adaptive update enables FebHD to provide a high quality of clustering with very few learning iterations. We also propose an end-to-end hardware accelerator that parallelizes the entire FebHD computation by exploiting FPGA bit-level granularity. Our evaluation shows that FebHD provides comparable accuracy to state-of-the-art clustering algorithms, while providing 6.2× and 9.1× (4.7× and 5.8×) faster and higher energy efficiency when running on the same FPGA (GPU) platform.**

I. Introduction

With the emergence of the Internet of Things (IoT), sensory and embedded devices generate massive data streams and demand services that pose substantial technical challenges due to limited device resources. Today's IoT applications analyze raw data by running machine learning algorithms [1]. Particularly in healthcare, the majority of data are not associated with any labels. This makes clustering algorithms the most popular learning methods for data analysis. 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]. 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.

Although there are several efforts on the acceleration of deep neural networks [3], there are much less focuses on real-time unsupervised learning. The process of clustering data involves heavy computations as most algorithms need to calculate pairwise distances between all the points in the dataset [4]. To give non-linearity to the clustering task, most existing algorithms use complex distance metrics, e.g., Euclidean distance, and use the iterative process for clustering. However, computing these distance metrics is extremely expensive on today's computers. In addition, the clustering algorithms require many costly iterations to converge to a reasonable solution.

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*. We exploit Hyperdimensional computing (HDC) [5], [6] as an alternative computing paradigm that emerged from theoretical neuroscience. HDC 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 [5], [7]. It thereby models human memory using points of a high-dimensional space. HDC is well suited to address learning tasks for healthcare systems as: (i) HDC models are computationally efficient and highly parallel at heart to train and amenable to hardware level optimization [8], [9], [10], (ii) HDC models offer an intuitive and human-interpretable model [11], [12], (iii) it offers a complete computational paradigm that can be applied to cognitive as well as learning problems [13], [14], [15], [16], [17], (iv) it provides strong robustness to noise – a key strength for IoT systems, and (v) it supports low-cost security and privacy [6], [18].

Although prior work tried to use HDC for efficient clustering [19], the lack of proper encoding and learning process results in low quality of learning even using a complex and costly cosine similarity metric. In this paper, we proposed FebHD, a novel framework for efficient and binary clustering in high-dimensional space. To the best of our knowledge, FebHD is the first fully binary HDC-based clustering approach that ensures robustness along with the same quality as the state-of-the-art clustering algorithms. The main contributions of this paper are listed in the following:

- FebHD introduces a novel encoding approach that maps data points into high-dimensional space while considering non-linear interactions that potentially occur between features of heterogeneous sensors. FebHD learning is performed by linear combinations of hypervectors that are non-linearly mapped to high-dimensional space. Our framework revisit clustering algorithms to ensure that the clustering algorithm is performing entirely over binary vectors. Particularly, FebHD simplifies the distance similarity, the most frequent operation used in clustering algorithms, to efficient and hardware-

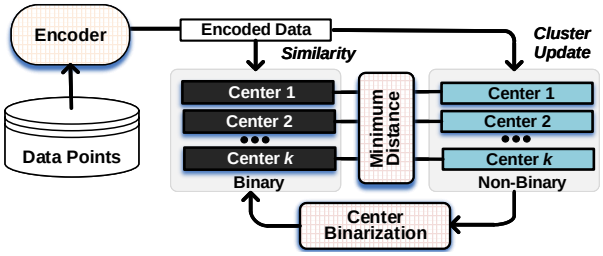

Fig. 1. Overview of FebHD clustering framework

friendly Hamming distance metric.

- We exploit HDC model transparency to enable FebHD to provide confidence about each prediction. FebHD uses this confidence level to adaptively update the cluster centers and provide fast convergence.
- We propose an end-to-end architecture supporting all essential FebHD operations on FPGA. Our implementation exploits the bit-level granularity of FPGAs to parallelize the key FebHD kernels such as Hamming distance computing and cluster updates using efficient lookup tables resources.

We evaluate FebHD efficiency on a wide range of clustering datasets. Our evaluation shows that FebHD provides comparable quality of clustering to the state-of-the-art clustering algorithms while providing 6.2× and 9.1× (4.7× and 5.8×) faster and higher energy efficiency when running on the same FPGA (GPU) platform. Our code is available open-source[1].

II. FebHD: Clustering in Binary Space

Figure 1 shows the overview of the FebHD framework. FebHD first maps data points into high-dimensional space. This encoding module needs to consider the relationship between different features and map data points into non-linear space where the similarity could be preserved using the Hamming metric. FebHD performs clustering over the encoded data. It generates k binary random hypervector representing k cluster centers. Then, it compares the Hamming distance similarity of each encoded data point with the cluster center. Each data point gets the label of a center that has the highest similarity with it. By going through the entire dataset, or a batch of data, FebHD assigns a label to each data point. FebHD updates the cluster centers depending on the confidence of the model on each prediction. The cluster update makes the centers to get non-binary elements. This eliminates using an efficient Hamming distance metric as a distance measurement in the next iterations. To address this, FebHD binarizes the new centers while still storing a copy of non-binary centers. The new binary centers are used for the distance similarity computation in order to assign a label to each data point. Depending on the new labels, FebHD updates the non-binary model. This iterative training continues until the number of labels does not change during a few iterations.

A. Non-Linear Encoding

There are multiple encoding methods proposed in literature [20], [21]. Although these methods have shown excellent classification accuracy for their application-specific problems, all existing encoding methods linearly combine the hypervectors corresponding to each feature, resulting in sub-optimal learning results. To obtain the most informative hypervectors, the HDC encoding should consider the non-linear interactions between the feature values with different weights.

In this context, we exploit the encoding method, proposed in [22], which exploits the kernel trick [23], [24] 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. Figure 1b shows our encoding procedure. Let us consider an encoding function that maps a feature vector $\vec{F} = \{f_1, f_2, \ldots, f_n\}$, with n features ($f_i \in \mathbb{N}$) to a hypervector $\vec{H}^b = \{h_1, h_2, \ldots, h_D\}$ with D dimensions ($h_i^b \in \{0,1\}$). We generate each dimension of the encoded data by calculating a dot product of the feature vector with a randomly generated vector as $h_i = \cos(\vec{B}_i \cdot \vec{F} + b_i) \times \sin(\vec{B}_i \cdot \vec{F})$, 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 HDC, binary (bipolar) hypervectors are often used for computation efficiency. Thus, our encoder has an option of binarizing the encoded hypervector with a sign function ($\vec{H}^b = sign(\vec{H})$).

B. Initial Center Generation

FebHD starts clustering from initial centers. The clustering starts by selecting randomly k encoded data points and using them as initial cluster centroids $\{\vec{H}_{i_1}, \vec{H}_{i_2}, \ldots, \vec{H}_{i_k}\}$, where $i_j \sim \mathrm{U}(1..n)$. Since these vectors are chosen from input data, each cluster will have at least one element assigned to it, which prevents empty cluster partitions.

C. Similarity Search

The next step of FebHD clustering is to assign each encoded data point to one of the cluster centers (Ⓑ). This is equivalent to unsupervised labeling of encoded data based on their similarity to cluster centers. Clustering algorithms are using different metrics for similarity search depending on the goal of clustering and representation of centers/data point. For example, K-means clustering often uses costly Euclidean distance as a similarity metric [25], [26]. Thanks to FebHD encoding (explained in Section II-A), the similarity metric can be simplified to cosine or Hamming distance. Prior work [19] showed that to get high quality of clustering, the values and centers need to represent using non-binary (e.g., integer or floating-point), requiring expensive cosine metric for distance measurement. The main goal of our proposed FebHD is to simplify the value's representation to binary and distance similarity metric to Hamming distance, with no impact on the clustering quality.

As Figure 1b shows, FebHD encoding module generates both non-binary ($\vec{H}$) and binary ($\vec{H}^b$) data. FebHD also stores two copies of the centers: (i) an original version of centers with non-binary representations ($\vec{C}_i$), and (ii) a binarized version of the cluster centers ($\vec{C}_i^b$). During the clustering, FebHD checks

[1] https://gitlab.com/biaslab/hd-clustering

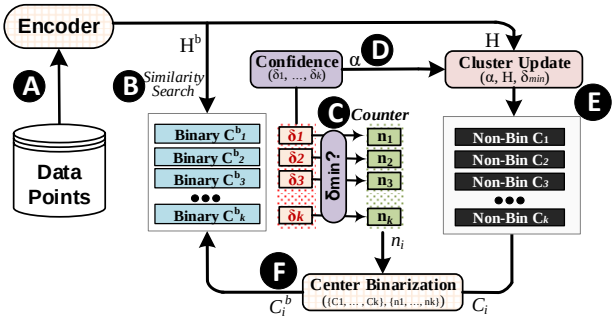

Fig. 2. FebHD framework supporting adaptive clustering.

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):

We use the same distance similarity measurement to assign all train data to the closest centers. FebHD also exploits HDC transparent model to give confidence level for each data point during the similarity search. This confidence rate specifies how close a data point is assigned to a center (D). For each data point, confidence defines as:

$$\alpha = \frac{\delta_{max} - \mu(\delta_j)}{\sigma(\delta_j)} \qquad 1 \leq j \leq k$$

Where δ_{max} is a cluster with the highest similarity with data, and $\mu(\delta_j)$ and $\sigma(\delta_j)$ show the average Hamming distance and standard deviation of the binary encoded data ($\vec{H}^b$) with other centers.

D. Adaptive Center Update

In the first iteration, FebHD provides a low quality of clustering as the initial cluster centers are often not a good representation of data. To enhance the clustering quality, we update the centers based on the new data point assigned to each center (E). Existing clustering approaches, e.g., K-means, compute the new centers by averaging over all training data point corresponding to each 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, FebHD proposes a method to find the new cluster centers based on the confidence that each data point has about the prediction.

FebHD Cluster Update: Each cluster center is updated using all data points assigned to the center as well as their corresponding confidence level. After assigning each encoding hypervector $\vec{H}$ of inputs belonging to center/label l, the center hypervector $\vec{C}_l$ can be obtained by bundling (adding) all $\vec{H}$s. Assuming there are J inputs having label l, the cluster update happens using one of the following options:

- *Baseline Non-Adaptive Update*: $\vec{C}_l \leftarrow \vec{C}_l + \sum_j^J \vec{H}_j$
- *Adaptive Non-binary Update*: $\vec{C}_l \leftarrow \vec{C}_l + \sum_j^J \alpha_j \vec{H}_j$
- *Adaptive Binary Update*: $\vec{C}_l \leftarrow \vec{C}_l + \sum_j^J \alpha_j \vec{H}_j^b$

Where $\vec{H}_i$ and $\vec{H}_i^b$ are non-binary and binary query data, respectively. All cluster updates are performed over the non-binary copy of the centers. Among the aforementioned methods, the adaptive cluster updates consider confidences value during the model update. The adaptive non-binary method uses an encoded query, while the adaptive binary update uses a binarized query for efficiency and hardware-friendly cluster update.

FebHD Cluster Binarization: The algorithm 1 describes FebHD clustering process using adaptive non-binary cluster update. Our algorithm takes binary and non-binary encoded hypervectors as inputs and returns each input to one of the k cluster centers. FebHD updates the binary centers based on the newly updated non-binary centers. This update aims to generate new binary centers that can better represent the cluster centers based on the distribution of the data. FebHD updates the binary cluster by simply binarizing every element 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 FebHD with binary representation, which is desired for hardware, we apply a *majority* function on non-binarized class hypervectors (F). Given a center hypervector, $\vec{C}_l = \langle c_D, \cdots, c_1 \rangle$, the majority function is defined as follows:

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

Using the majority function, the final hypervector for each data point is encoded by $\vec{C}^b = MAJ(\vec{C}_l, n_i/2)$, 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

FebHD performs an iterative process for clustering. In each iteration, FebHD computes binary data similarity with the binary centers and then updates the non-binary centers depending on the search results and the search confidence. Then, we binarize the updated centers in order to find new binary centers. After every iteration, FebHD checks the changes in the cluster centers in order to decide the convergences. FebHD is converged when all binary cluster centers have less than $\beta\%$ changes in their elements during a few consecutive iterations, where β is a convergence threshold ranging from 0% to 5%. Our solution enables each clustering iteration to go over the entire or a batch of data. A large batch size provides a 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 FebHD computation by reducing the number of iterations. In section IV-E, we explore the impact of batch size on FebHD efficiency and quality of clustering.

III. FebHD FPGA Acceleration

FebHD can be implemented in different platforms such as CPU, GPU, or FPGA. Due to many bitwise operations involved in FebHD, FPGA is a suitable candidate for hardware acceleration. 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 a binarized projection matrix that simplifies

Algorithm 1 FebHD clustering process

function FEBHD($\vec{H}, \vec{H}^b, k$)
  $\vec{C}_i \leftarrow Random\{\vec{H}_1, \ldots, \vec{H}_n\}$ ▷ Compute initial centers
  $\vec{C}^b_i \leftarrow sign(\vec{C}_i)$ ▷ Binary centers
  for $it = 1$ **to** `maxiter` **do**
    for $i = 1$ **to** n **do**
      $\delta_j \leftarrow \delta(\vec{C}^b_j, \vec{H}^b_i)$ ▷ For $1 \le j \le k$
      $\delta_{max} \leftarrow \max_{1 \le j \le K} \delta_j$
      $\alpha_i \leftarrow (\delta_{\max} - \mu(\delta_j))/\sigma(\delta_j)$
      $\vec{C}_{\delta_{\max}} \leftarrow \vec{C}_{\delta_{\max}} + \alpha_i \vec{H}_i$
      $\vec{C}^b_{\delta_{\max}} \leftarrow \text{sign}(\vec{C}_{\delta_{\max}})$ ▷ Update model
    end for
    if No Change on Centers? **then** ▷ Converge check
      return
    end if
  end for

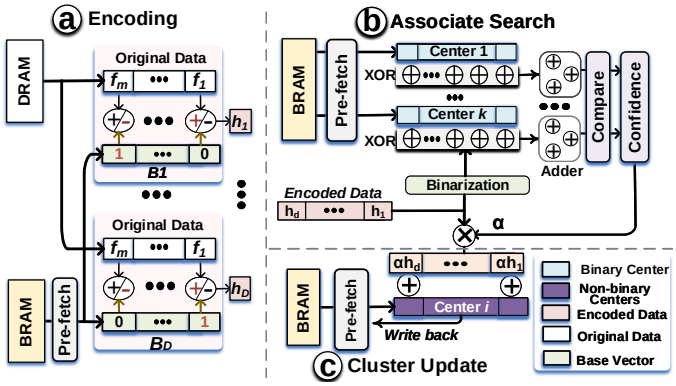

Fig. 3. FebHD FPGA implementation.

the encoding task to the addition and subtraction of different input features. As Figure 3a shows, our implementation reads m features of original data from DRAM. Due to the 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 $\{\vec{B}_1, \cdots, \vec{B}_D\}$ 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 encodes them based on the base values in those new dimensions. Depending on the number of features or dimensionality of the encoded 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 the 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 centers. Our implementation accumulates the `XOR` results over D dimensions in order to find a Hamming distance of data with each center. Finally, we compute the confidence level depending on all k Hamming distance values; each corresponds to a cluster center. Thanks to the FebHD similarity metric, the associative search is implemented using FPGA LUTs in a highly parallel and efficient way.

C. Cluster update

For cluster update, FebHD keeps two copies of the centers in BRAM blocks: a non-binary and a binary. In contrast to the 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.

Our implementation works in a pipeline to maximize resource utilization. When FPGA encodes i^{th} data, the associative search computes the similarity of the $i-1^{th}$ data with the centers. Simultaneously, FPGA updates one of the non-binary centers based on the similarity search of the $i-2^{th}$ data.

TABLE I
DATASETS

	Data	*Features*	*Clusters*	*Description*
DIM	1024	128	16	Gaussian Data in high-dimension [27]
A3	7500	2	50	Low-dimensional distributed data [28]
TETRA	236	16	9	Genetic tetragonula bees [29]
MNIST	70000	784	10	Handwritten digit recognition [30]
VERONICA	207	586	2	Genetic AFLP of Veronica plants [31]
UCIHAR	10299	561	6	Human Activity Recognition [32]
SYNTHET I	1000	100	25	Synthetic Data
SYNTHET II	100000	100	25	Synthetic Data

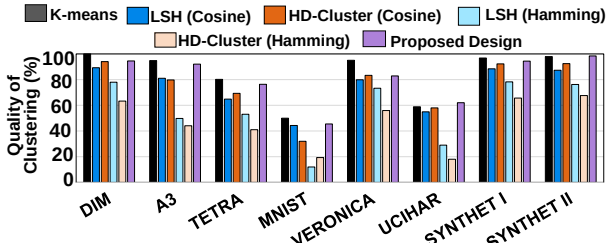

Fig. 4. Comparison of FebHD quality of clustering with other algorithms.

IV. Evaluation

A. Experimental Setup

We used C++ software implementation for FebHD learning and verification. We exploit the Scikit-learn library [33] for implementing of the state-of-the-art clustering approaches. For hardware support, we fully implemented FebHD in RTL using Verilog HDL. FebHD is deeply pipelined to run with 200 MHz clock frequency. We verify the timing and functionality of the FebHD using both synthesis and real implementation on Xilinx Vivado Design Suite [34]. To estimate the FPGA's power consumption, we use the builtin Xilinx Power Estimation tool in the Vivado Design Suite. The results are reported for Kintex-7 FPGA. We examine FebHD efficiency and quality of clustering on several large-scale datasets including actual and synthetic datasets. Table I lists all popular datasets. 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.

B. FebHD vs. Other Clustering Algorithms

Figure 4a compares FebHD quality of clustering with k-means and the state-of-the-art clustering approaches in high-dimensional space: HDC-based clustering (HD-cluster) [19], and clustering based on Locality Sensitive Hashing (LSH-cluster) [35]. K-means algorithm is working on original data and uses 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

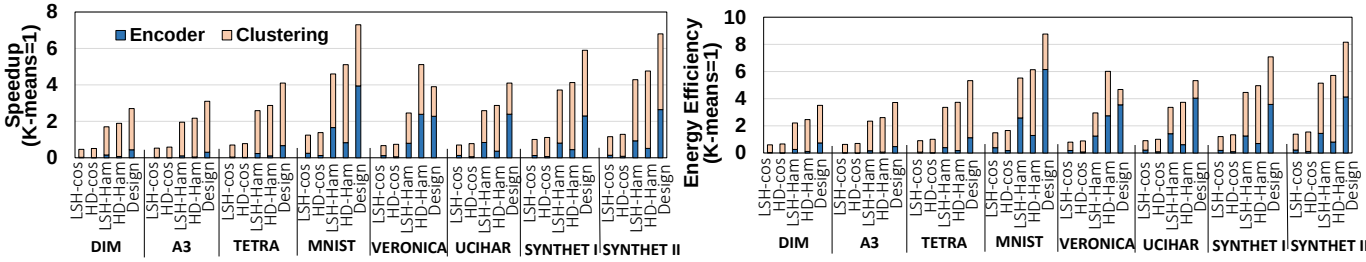

Fig. 5. FebHD computation efficiency as compared to other clustering algorithms.

results are reported using both cosine and Hamming distance metrics. The cosine metric is used for vectors with integer precision and Hamming metric for a case of binary vectors. Our evaluation shows that FebHD provides comparable quality of clustering to k-means. In addition, as compared to HD-cluster and LSH-cluster (using cosine metric), FebHD provides a significantly higher quality of the clustering. Compared with HD and LSH-based approaches with the Hamming metric, which have similar computation complexity as FebHD, our approach provides 24.3% and 33.5% higher quality of clustering. Even using the cosine metric, HD-cluster and LSH-cluster provide 7.5% and 8.9% lower quality of clustering as compared to FebHD. 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 each prediction confidence. The clustering quality metric used was the Adjusted Mutual Information (AMI) score, which is often used for this purpose [36].

C. FebHD Efficiency

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

FebHD also provides higher efficiency than both LSH-Hamming and HD-Hamming. FebHD uses more complex encoding than both LSH and HD-based clustering approaches. As Figure 5 shows, FebHD encoding takes 36% of total energy, while this portion is less than 20% and 11% in LSH-Hamming and HD-Hamming. Therefore, in terms of a single iteration, FebHD consumes about 1.2× and 1.3× higher energy than LSH-Hamming and HD-Hamming. However, FebHD adaptive clustering significantly reduces the number of required clustering iterations. The number of iterations has a direct impact on clustering efficiency. Our evaluation shows that FebHD provides, on average, 1.6× and 1.4× higher energy efficiency as compared to LSH-Hamming and HD-Hamming, respectively.

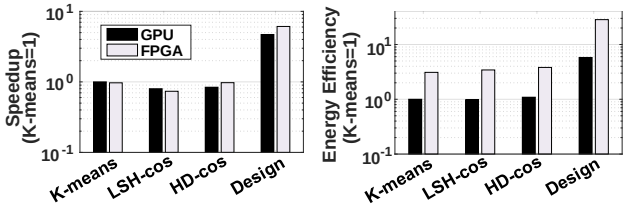

Fig. 6. FebHD efficiency on GPU vs FPGA.

Figure 6 compares FebHD 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 [37], which fully utilizes FPGA to maximize the throughput. For fairness, we report HD and LSH-based clustering only for cosine metric (LSH-cos, and HD-cos), when they provide comparable accuracy to FebHD. Our evaluation shows that the algorithm using non-binary representation has minor efficiency improvement on FPGA, especially in terms of execution time. Implementing the cosine metrics requires utilizing the limited FPGA DSPs, resulting in low computation efficiency. In contrast, FebHD can perform the 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 FebHD on FPGA can provide 6.2× and 9.1× (6.0× and 7.5×) faster and higher energy efficiency as compared to K-means (HD-cosine) running on FPGA.

D. FebHD & adaptive cluster Update

We compare FebHD quality of clustering using different cluster update methods introduced in Section II-D: (i) non-adaptive update, (ii) adaptive non-binary update, and (iii) adaptive binary update. Table II compares FebHD in these configurations from the quality of clustering and the number of required iterations to converge. Our evaluation shows that FebHD with non-adaptive update gives all data points the same chance regardless of their prediction confidence. This increases the number of requires the number of required iterations for clustering. The adaptive update improves the quality and efficiency of the clustering. Although FebHD with

TABLE II
FEBHD 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

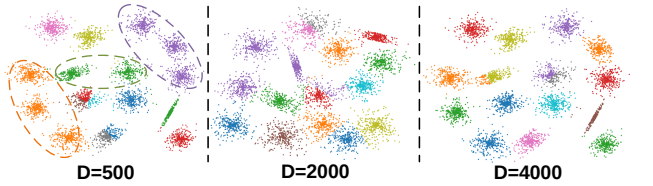

Fig. 7. FebHD visual clustering results in different dimensions.

adaptive binary update improves the efficiency of a single learning iteration, the binary update requires more iterations to converge, resulting in lower overall efficiency as compared to FebHD using adaptive non-binary update.

Figure 7a visualizes FebHD quality of clustering during different dimensions. The results are reported for FebHD using three different dimensions. As the figure shows, FebHD quality of clustering improves by increasing dimensionality.

E. FebHD Batch Size Trade-offs

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

V. CONCLUSION

In this paper, we proposed FebHD, an adaptive framework for efficient and fully binary clustering in high-dimensional space. FebHD performs clustering by assigning each data point to a set of initialized centers in high-dimensional space. It then updates the centers adaptively based on: (i) data points assigned to each cluster, and (ii) the confidence of the model on each prediction. We also propose an end-to-end hardware accelerator that parallelizes the entire FebHD computation by exploiting FPGA bit-level granularity.

ACKNOWLEDGMENT

This work was partially supported by Semiconductor Research Corporation (SRC) Task No. 2988.001. Mohsen Imani and Yeseong Kim are co-corresponding authors of the paper.

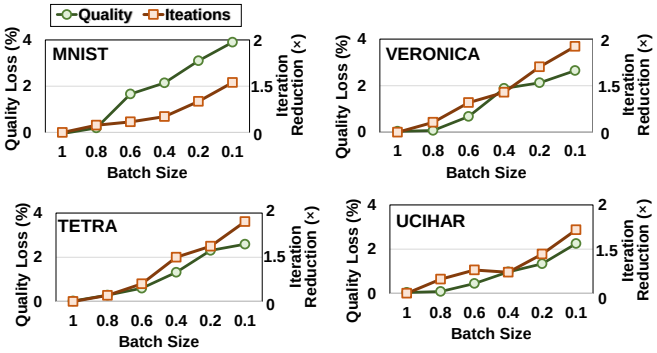

Fig. 8. Impact of the batch size on FebHD quality and convergence.

REFERENCES

[1] A. M. Rahmani, P. Liljeberg, J.-S. Preden, and A. Jantsch, *Fog computing in the internet of things: Intelligence at the edge*. Springer, 2017.
[2] A. K. Jain, "Data clustering: 50 years beyond k-means," *Pattern recognition letters*, vol. 31, no. 8, pp. 651–666, 2010.
[3] P. Yin, C. Wang, H. Waris, W. Liu, Y. Han, *et al.*, "Design and analysis of energy-efficient dynamic range approximate logarithmic multipliers for machine learning," *T-SUSC*, 2020.
[4] L. Fu *et al.*, "Cd-hit: accelerated for clustering the next-generation sequencing data," *Bioinformatics*, vol. 28, no. 23, pp. 3150–3152, 2012.
[5] 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.
[6] M. Imani *et al.*, "A framework for collaborative learning in secure high-dimensional space," in *CLOUD*, pp. 435–446, IEEE, 2019.
[7] M. Imani *et al.*, "A binary learning framework for hyperdimensional computing," in *DATE*, pp. 126–131, IEEE, 2019.
[8] M. Imani *et al.*, "Exploring hyperdimensional associative memory," in *HPCA*, pp. 445–456, IEEE, 2017.
[9] M. Imani *et al.*, "Revisiting hyperdimensional learning for fpga and low-power architectures," in *HPCA*, IEEE, 2021.
[10] M. Imani *et al.*, "Quanthd: A quantization framework for hyperdimensional computing," *IEEE TCAD*, 2019.
[11] 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.
[12] M. Imani *et al.*, "Semihd: Semi-supervised learning using hyperdimensional computing.," in *ICCAD*, pp. 1–8, 2019.
[13] M. Nazemi *et al.*, "Synergiclearning: Neural network-based feature extraction for highly-accurate hyperdimensional learning," *arXiv preprint arXiv:2007.15222*, 2020.
[14] M. Imani *et al.*, "Bric: Locality-based encoding for energy-efficient brain-inspired hyperdimensional computing," in *IEEE/ACM DAC*, pp. 1–6, 2019.
[15] Y. Kim *et al.*, "Efficient human activity recognition using hyperdimensional computing," in *IOT*, p. 38, ACM, 2018.
[16] S. Salamat *et al.*, "Accelerating hyperdimensional computing on fpgas by exploiting computational reuse," *TC*, 2020.
[17] Y. Kim *et al.*, "Geniehd: Efficient dna pattern matching accelerator using hyperdimensional computing," in *DATE*, IEEE, 2020.
[18] B. Khaleghi *et al.*, "Prive-hd: Privacy-preserved hyperdimensional computing," *arXiv preprint arXiv:2005.06716*, 2020.
[19] M. Imani *et al.*, "Hdcluster: An accurate clustering using brain-inspired high-dimensional computing," in *DATE*, pp. 1591–1594, IEEE, 2019.
[20] M. Imani *et al.*, "Hierarchical hyperdimensional computing for energy efficient classification," in *DAC*, p. 108, ACM, 2018.
[21] A. Rahimi *et al.*, "A robust and energy-efficient classifier using brain-inspired hyperdimensional computing," in *ISLPED*, pp. 64–69, 2016.
[22] M. Imani *et al.*, "Dual: Acceleration of clustering algorithms using digital-based processing in-memory," in *IEEE/ACM MICRO*, pp. 356–371, IEEE, 2020.
[23] A. Rahimi *et al.*, "Random features for large-scale kernel machines," in *NIPS*, pp. 1177–1184, 2008.
[24] B. Schölkopf, "The kernel trick for distances," in *Advances in neural information processing systems*, pp. 301–307, 2001.
[25] N. Dhanachandra *et al.*, "Image segmentation using k-means clustering algorithm and subtractive clustering algorithm," *Procedia Computer Science*, pp. 764–771, 2015.
[26] H. Koga *et al.*, "Fast agglomerative hierarchical clustering algorithm using locality-sensitive hashing," *KAIS*, vol. 12, no. 1, pp. 25–53, 2007.
[27] P. Franti *et al.*, "Fast agglomerative clustering using a k-nearest neighbor graph," *TPAMI*, vol. 28, no. 11, pp. 1875–1881, 2006.
[28] I. Kärkkäinen and P. Fränti, *Dynamic local search algorithm for the clustering problem*. University of Joensuu Joensuu, Finland, 2002.
[29] P. Franck *et al.*, "Nest architecture and genetic differentiation in a species complex of australian stingless bees," *Molecular Ecology*, vol. 13, no. 8, pp. 2317–2331, 2004.
[30] Y. LeCun *et al.*, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.
[31] M. M. Martínez-Ortega *et al.*, "Species boundaries and phylogeographic patterns in cryptic taxa inferred from aflp markers: Veronica subgen.," *Systematic Botany*.
[32] D. Anguita *et al.*, "Human activity recognition on smartphones using a multiclass hardware-friendly support vector machine," in *AAL*, pp. 216–223, Springer, 2012.
[33] F. Pedregosa *et al.*, "Scikit-learn: Machine learning in python," *JMLR*, vol. 12, no. Oct, pp. 2825–2830, 2011.
[34] T. Feist, "Vivado design suite," *White Paper*, vol. 5, 2012.
[35] X. Shen *et al.*, "Compressed k-means for large-scale clustering," in *AAAI*, 2017.
[36] N. X. Vinh *et al.*, "Information theoretic measures for clusterings comparison," in *ICML*, ACM Press, 2009.
[37] Z. He *et al.*, "Bis-km: Enabling any-precision k-means on fpgas," in *FPGA*, pp. 233–243, 2020.