HD2FPGA: Automated Framework for Accelerating Hyperdimensional Computing on FPGAs

Tinaqi Zhang*, Sahand Salamat*, Behnam Khaleghi*, Justin Morris†, Baris Aksanli‡, Tajana Simunic Rosing*
* UC San Diego; La Jolla, CA 92093, USA
† California State University, San Marcos; San Marcos, CA 92096, USA
‡ San Diego State University; San Diego, CA 92182, USA
Email: *{tiz014, sasalama, bkhaleghi, tajana}@ucsd.edu, †{justinmorris}@csusm.edu, ‡{baksanli}@sdsu.edu,

2023 24th International Symposium on Quality Electronic Design (ISQED) | 979-8-3503-3475-3/23/$31.00 ©2023 IEEE | DOI: 10.1109/ISQED57927.2023.10129332

***Abstract*—Building a highly-efficient FPGA accelerator for Hyperdimensional (HD) computing is tedious work that requires Register Transfer Level (RTL) programming and verification. An inexperienced designer might waste significant time finding the best resource allocation scheme to achieve the target performance under resource constraints, especially for edge applications. HD computing is a novel computational paradigm that emulates brain functionality in performing cognitive tasks. The underlying computations of HD involve a substantial number of element-wise operations (e.g., additions and multiplications) on ultra-wide hypervectors (HVs), which can be effectively parallelized and pipelined. Although different HD applications might vary in terms of the number of input features and output classes (labels), they generally follow the same computation flow. In this paper, we propose HD2FPGA, an automated tool that generates fast and highly efficient FPGA-based accelerators for HD classification and clustering. HD2FPGA eliminates the arduous task of hand-crafted design of hardware accelerators by leveraging a template of optimized processing elements to automatically generate an FPGA implementation as a function of application specifications and user constraints. For HD classification HD2FPGA, on average, provides $1.5\times$ (up to $2.5\times$) speedup compared to the state-of-the-art FPGA-based accelerator and $36.6\times$ speedup with $5.4\times$ higher energy efficiency compared to the GPU-based one. For HD clustering, HD2FPGA is $2.2\times$ faster than the GPU framework.**

***Index Terms*—Hyperdimensional (HD) computing, Automation, FPGA**

I. Introduction

Hyperdimensional (HD) computing imitates brain functionalities when performing cognitive tasks with low-energy consumption, making it popular in energy-bounded applications [1]. Researchers explored implementing HD models on FPGAs to obtain hardware-level energy savings [2], [3], [4], [5], but designing FPGAs is a time-consuming process [6]. Besides, most of those FPGA designs are customized by specific tasks or datasets with lots of assumptions. The emerging applications of HD motivate researchers to develop flexible frameworks to lower the bar of deploying HD algorithms and to accelerate the development cycle. The work in [7] proposed a GPU-powered generic implementation of HD. However, GPUs generally have high power consumption. The work in [5] proposed an automated framework to accelerate HD classification on FPGA aiming to accelerate FPGA development cycle. However, it only supports classification and is based on several assumptions which may not be applicable to all applications. In this work, we develop a framework with a user-friendly GUI that can automatically generate FPGA designs both for HD clustering and classification problems, and it requires a smaller number of dimensions to achieve comparable accuracy with the state of the art [5].

HD computing is based on human brain operations where the brain computes with patterns of neural activity, which can be realized by *encoding* data into high-dimensional vectors, called hypervectors (HVs) with thousands of bits. Many machine learning problems have been implemented using HD computing, including classification and clustering [8], [3], [9]. HD computing is able to match the accuracy of state-of-the-art machine learning algorithms while learning from only a small portion of training data [10], [11]. Besides, HD computing is inherently robust against noise due to its large vector dimensionality [1].

HD2FPGA automatically generates an FPGA-based accelerator for HD classification and clustering to abstract away the implementation complexities and long design cycles associated with hardware design from the user. It reduces the time of implementing the HD models from weeks down to hours. HD2FPGA generates a synthesizable Verilog implementation of an HD accelerator while taking high-level user input and target FPGA parameters into account. It is flexible and highly optimized to deliver a fast and an energy-efficient accelerator according to user-specified constraints (i.e., performance, and power). HD2FPGA supports end-to-end training, retraining, and inference for both HD classification and clustering problems. Specifically, HD2FPGA makes the following contributions, where it:

- Develops a user-friendly framework that generates FPGA-based synthesizable architectures for accelerating training, retraining, and inference for HD classification and clustering problems.
- Utilizes random projection encoding to deliver high accuracy for multiple problems while removing the dependency of the encoding hardware on the number of input features.
- Enables easy access for researchers and industry to implement classification and clustering on FPGAs with orders of magnitude speed-up and energy reduction, compared to CPU and GPU, with a push of a button by using HD2FPGA's Graphical User Interface.
- Evaluates the efficiency of HD2FPGA classification

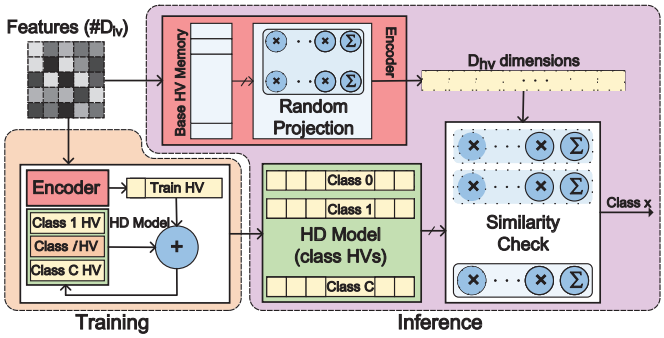

Fig. 1: Overview of hyperdimensional learning and inference.

and clustering using different benchmarks. Compared to the state-of-the-art FPGA-based HD framework [5], HD2FPGA provides $4.2\times$ speedup in HD training and $1.5\times$ speedup in HD classification. Compared to the state-of-the-art GPU-based HD framework [7], HD2FPGA provides $36.6\times$ speedup and $5.4\times$ higher energy efficiency in HD classification with comparable accuracy, and $2.2\times$ speedup in HD clustering. Compared to running HD on a CPU, HD2FPGA accelerates training HD classification by $30.5\times$ and increases energy efficiency by $73.9\times$. HD2FPGA shows $578\times$ ($57\times$) speed up and $1503\times$ ($140\times$) higher energy efficiency in HD classification (clustering), respectively.

II. Background and Related Work

In this section, we first articulate the operations behind HD computing, including encoding, training, inference, and retraining for classification and clustering. Then, we review the previous work on HD computing.

A. Hyperdimensional Computing

HD computing builds on the fact that the cognitive tasks of the human brain can be explained by mathematical operations on ultra-wide HVs [1]. In other words, the brain computes with patterns of neural activity, which can be better represented by HVs rather than scalar numbers. An HV comprises $\mathcal{D}_{hv}$, e.g., 10,000 bits, independent components (dimensions) whereby the enclosed information is distributed uniformly among all $\mathcal{D}_{hv}$ dimensions. This makes HVs robust to failure as the system remains functional under a certain number of component failings, and as degradation of information does not depend on the position of the failing components [2], [8].

Encoding: In HD2FPGA we utilize the random projection encoding for classification and clustering. Suppose the input data is $vecV = \{v_1, v_2, \ldots, v_{D_{iv}}\}$, where D_{iv} is the number of input features in the input vector and each feature can be quantized to L unique levels. In this encoding, the input vector is multiplied by a projection matrix consisting of a set of D_{iv} binary vectors P_i. The P matrix consists of D_{iv} columns of projection HVs (P_i). Each P_i consists of D_{hv} randomly generated binary values.

$$P_i = \{p_j\}_{j=1}^{j=D_{hv}},\ p_j \in \{0,1\} \tag{1}$$

In the random projection encoding, the input bit-width only affects the operations bit-width. The projection HVs should be nearly orthogonal and it has been shown that applying circular shift operations on base HVs generates nearly orthogonal HVs needed to represent each input dimension [3], [5]. After loading the projection HVs, the projection matrix is getting multiplied by the input features to generate the encoded HV. The encoded vector $\vec{H} = \{h_i\}_{i=1}^{i=D_{hv}}$ can be represented as:

$$\vec{H} = P \times \vec{V} \tag{2}$$

Training: After mapping each input $\vec{V}_{iv}$ to HV $\vec{H}$ as above, all HVs belonging to the same class (label) are simply summed to form the final representative HVs. Thus, assuming $\vec{H}^l = \langle h_0, h_1, \cdots, h_{d_{hv}} \rangle^l$ denotes a generated class HV for an input data with label l, the final (representative) class HVs are obtained as Equation 3, in which each dimension c_j is obtained through dimension-wise addition of all h_j^ls, and $\mathcal{K}$ is the number of input data with label l.

$$\vec{C}_l = \langle c_0, c_1, \cdots, c_{d_{hv}} \rangle = \sum_{k=0}^{\mathcal{K}} \vec{H}_k^l \tag{3}$$

Inference: The first steps of inference in HD computing are similar to training; an input feature vector is encoded to $\mathcal{D}_{hv}-$dimension query HV, as explained in the encoding step. This is followed by a similarity check between the query HV $\vec{\mathcal{H}}$ and all representative class HVs, $\vec{\mathcal{C}}_l$. The similarity is defined as calculating the cosine similarity, which is obtained by multiplying each dimension in the query vector by the corresponding dimension of the class HVs, and adding up the partial products:

$$\text{similarity}(\vec{\mathcal{H}}, \vec{\mathcal{C}}_l) = \sum_{j=0}^{\mathcal{D}_{hv}} h_k \cdot c_k \tag{4}$$

The class with the highest similarity with the query HV indicates the classification result.

Retraining: Retraining might be used to enhance the model accuracy by calibrating it either via new training data or by multiple iterations on the same training data. Retraining is basically done by removing the mispredicted query HVs from the mispredicted class and adding it to the right class. Thus, for a new input feature vector $\vec{V}_{in}$ with query HV $\vec{\mathcal{H}}$ belonging actually to class with HV $\vec{\mathcal{C}}_l$, if the current model predicts the class $\mathcal{C}_{l'}$ where $\mathcal{C}_{l'} \neq \mathcal{C}_l$, the model updates itself as follows:

$$\begin{aligned} \vec{\mathcal{C}}_l &= \vec{\mathcal{C}}_l + \vec{\mathcal{H}} \\ \vec{\mathcal{C}}_{l'} &= \vec{\mathcal{C}}_{l'} - \vec{\mathcal{H}} \end{aligned} \tag{5}$$

This, indeed, reduces the similarity between $\vec{\mathcal{H}}$ and mispredicted class $\mathcal{C}_{l'}$, and adds $\vec{\mathcal{H}}$ to the correct class $\mathcal{C}_l$ to increase their similarity and the model will be able to correctly classify such query HVs.

Clustering: Similar to HD classification, in HD clustering every input needs to be encoded first. In HD clustering, the centroid HVs are initialized by the HV of randomly selected inputs. HD clustering keeps 2 copies of the HD model

(centroid HVs). It uses one for finding the closest centroid to the current input, and uses the other model to update the clustering centroids for the next iteration. During each iteration of clustering, it finds the similarity between the encoded HV and every centroid, and the cluster with the maximum similarity is the result of clustering (similar to classification). After finding the closest cluster, the input is used to update the copy of HD model. The encoded input is added to the centroid HV representing the cluster. After processing all the input in the dataset, the HD model is replaced by the updated model for the next iteration. The number of iterations is a parameter defined by the user.

B. Related Work

HD computing is gaining traction as an alternative solution to perform cognitive tasks in a light-weight fashion. It uses significantly simpler operations compared to conventional machine learning techniques that deal with complex learning procedures with a substantial number of costly operations. HD computing has been used for a wide range of applications, including classification [12], clustering [9], recommendation systems [13], etc.

Several studies have attempted to propose application-specific accelerators (ASIC [14], in-memory computing[15]), and FPGA [2], [3], [4], [5] to enhance the efficacy of HD computing.

The work in [15] presented a complete in-memory platform for executing both encoding and associative steps of HD computing on memristive devices. Thanks to the intrinsic robustness of HD computing to noise, the work in [15] approximates the mathematical operations to further accelerate HD computing. However, those in-memory computing solutions are not able to support flexible encoding bit-length, which limits the application to specific datasets and encoding methods though shows astonishing energy efficiency. The FPGA designs, in contrast, can provide more flexibility to designer in making trade-offs between metrics. The study in [2] proposes an approximate majority gate to compose the binary class HVs without requiring to hold the summation on HV components in a multi-bit format in the course of training. This is, however, limited to low-accuracy binarized HD computing. The work in [3] exploits computational reuse to reduce the computation complexity of HD classification and is implemented on FPGA for classification. It reuses the previously encoded HVs to encode the current input to simplify the encoding step. It additionally, clusters class HVs dimensions to reduce the number of multiplications in the associative search step. The work in [4] proposes approximate encoding modules to accelerate the encoding step. They approximate and optimize the encoding operations based on the characteristics of FPGA resources. However, these designs require lots of time to implement HD models for different applications on FPGAs [6].

In F5HD [5], an automated framework to implement HD classification on FPGA is implemented. F5HD only supports inputs quantized to 8 values (3 bits) which damages the accuracy in many applications. In this work, we develop a framework that can automatically generate FPGA program for both HD classification and HD clustering. Additionally, HD2FPGA utilizes random projection encoding to provide higher accuracy and bring more flexibility in supporting various applications. HD2FPGA supports 32-bit input features to provide higher applicability in supporting various applications. HD2FPGA is also equipped with a GUI to facilitate the implementation of HD classification and clustering on FPGA for users.

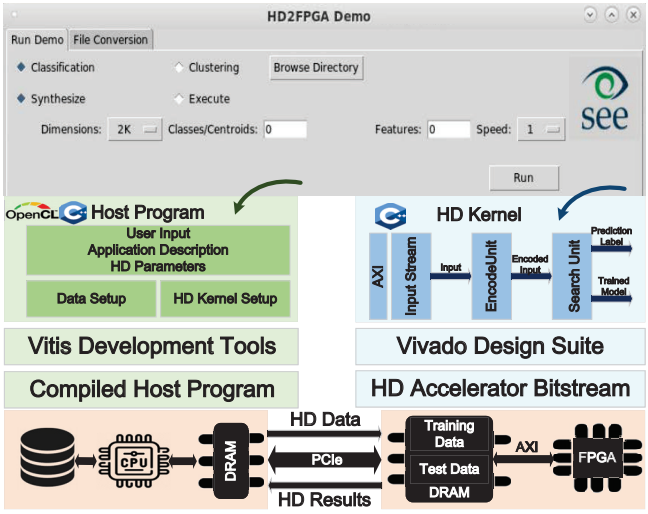

Fig. 2: Overview of the proposed framework, HD2FPGA.

III. HD2FPGA Framework

HD2FPGA aims to abstract away the complexities behind employing FPGAs for accelerating AI applications. HD2FPGA is an automated tool to accelerate the design time of an FPGA-based HD classification and clustering considering the user-specified criteria, e.g., power budget, performance-accuracy trade-off, and FPGA model (available resources). The overview of HD2FPGA is illustrated in Figure 2. HD2FPGA can be split into two parts: the host program and the HD kernel. The host program runs on the host CPU and is responsible for transferring data to the FPGA off-chip memory, initiating the HD kernel, and reading the results from the FPGA. HD2FPGA gets the application description and HD parameters from the user and automatically generates an FPGA-based accelerator, called HD kernel, for the user's application. The HD kernel runs on FPGA and supports end-to-end HD execution, including encoding, training, retraining, and inference of HD classification and clustering.

A. HD2FPGA Architecture

HD2FPGA automatically generates an FPGA-based accelerator for HD classification and clustering applications. HD2FPGA gets the HD parameters (e.g., HV lengths, number of input features, number of classes/clusters) and generates an accelerator based on user inputs. HD2FPGA consists of three main modules (InputStream, EncodeUnit, and SearchUnit) to

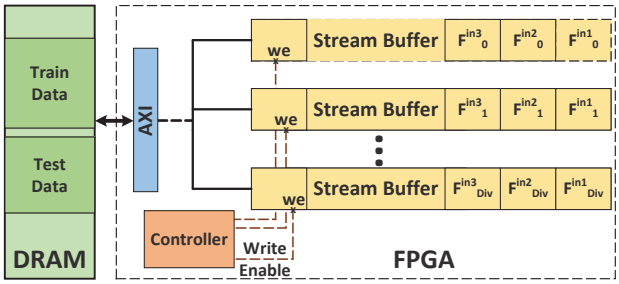

Fig. 3: The architecture of the InputStream module.

execute all the operations in both HD classification and clustering. Details of each module are discussed in the following.

InputStream: It reads the inputs from the FPGA memory through AXI bus interfaces and writes them into parallel buffers as illustrated in Figure 3.The number of features for each input is a parameter (D_{iv}) set by the user, during the synthesis. The InputStream module continuously reads the D_{iv} input features and writes them into parallel stream buffers, implemented as First-In-First-Out (FIFO) buffers, as shown in Figure 3. Features belonging to an input vector can be read in parallel as they are written in different stream buffers. F_i^j, in the figure, represents the i^{th} feature of the j^{th} input. The encoding module needs to read multiple input features in each cycle, therefore, parallel buffers provide simultaneous access to all input features.

EncodeUnit: It reads the input features from the stream buffers and encodes the input to an HV with D_{hv} dimensions (defined by the user during the runtime). HD2FPGA utilizes random projection encoding following equation 2, as it is more efficient for edge devices and provides higher accuracy in most applications. The input vector $\vec{v}$ shows a quantized feature element, while P_k is the k^{th} randomly generated projection HV. Projection HVs are frequently used to generate the encoded HVs. Thus, they should be stored in FPGA on-chip memory (BRAMs and/or URAMs) to maximize performance. To make HD2FPGA compatible with more FPGA family types and specifically smaller FPGAs used in edge devices with limited BRAMs, the HD kernel only stores a single seed HV (SeedHV), and by applying permutations, it doesn't store the projection HV and creates them on the fly. To permute the seed HV, HD2FPGA uses hardware-friendly rotational shift operation as shown in the following equation to generate the projection HVs.The random projection encoding by using the seed HV is shown in the equation below.

$$H = \sum_{k=1}^{D_{iv}} P_k \times v_k = \sum_{k=1}^{D_{iv}} (seedHV << k) \times v_k \qquad (6)$$

Figure 4(a) shows the matrix multiplication operations for the random projection encoding. B_i is the i^{th} dimension of the seed HV and column P_i is the projection HV of the i^{th} feature. P_1 directly uses the seed HV, and the other projection HVs are conducted by permuting P_1. HD2FPGA breaks down the encoding operations to generate the encoded HV in multiple cycles to be compatible with a wider range of FPGA devices with different amounts of resources. HD2FPGA is able to increase or decrease the parallelization based on the users' desired performance and the available FPGA resources. It accelerates HD operations by parallelizing the HD operations at the dimension level. Each dimension of the encoded HV can be generated independently of the others. The similarity metrics between dimensions of the encoded HV and class HVs can also be calculated independently of the other dimensions.

HD2FPGA uses two parameters to adjust the parallelization of HD to make a trade-off between parallelization and resource utilization. It breaks down the matrix multiplication computation using a sliding window that covers C elements of the input features and generates R dimensions of the encoded HV in $\frac{D_{iv}}{C}$ cycles in parallel. To be specific, at each cycle, HD2FPGA multiplies the C columns R rows of the projection binary vectors with the corresponding C elements of the features to generate a portion of a dimension of the encoded HV. And then, the window slides to the next portion of the same rows to generate the remaining partials. As the window slides over rows, it also slides on the features column so we are always multiplying elements of column k with the element k of the feature column. This procedure accumulates the partial results of the same encoding dimensions over several cycles, without moving the partial accumulation results. Thus, we have a fixed architecture consisting of several pipelined tree-adders for vector-vector multiplications in which we just need to feed the proper dimensions (bits) of the projection HVs and feature values to tree-adders, in a deterministic sequence, as shown in Figure 4(b).

The generated dimensions of the encoded HV are written into a memory, which is used for the following HD steps, including training and associative search. The HD kernel stores the entire encoded HV for the retraining step in the off-chip memory to avoid re-encoding. During the retraining steps, the encoded HVs are fetched to adjust the HD model for any misprediction. Thus, in the retraining steps, encoding is not executed from scratch anymore.

SearchUnit: It executes HD training, inference, and retraining. If the user loads a trained HD model, SearchUnit loads the trained class HVs from the FPGA memory. Otherwise, the class HVs are initialized with zeros, and the SearchUnit updates the HD model during the training. To train the model, it adds the encoded HVs to the class HV they belong to. The number of retraining iterations is empirically set by the user during the runtime such that the difference between the accuracy of the HD model between two consecutive iterations is less than the user's desired threshold (the model converges). In the retraining step, the predicted label is compared to the original label of the data. In case of misprediction, SearchUnit adds the encoded HV to the class HV it belongs to and subtracts the encoded HV from the predicted class. At the end of retraining, SearchUnit writes the trained HD model into the FPGA memory. To perform HD inference, SearchUnit calculates the cosine similarity between the encoded HV and

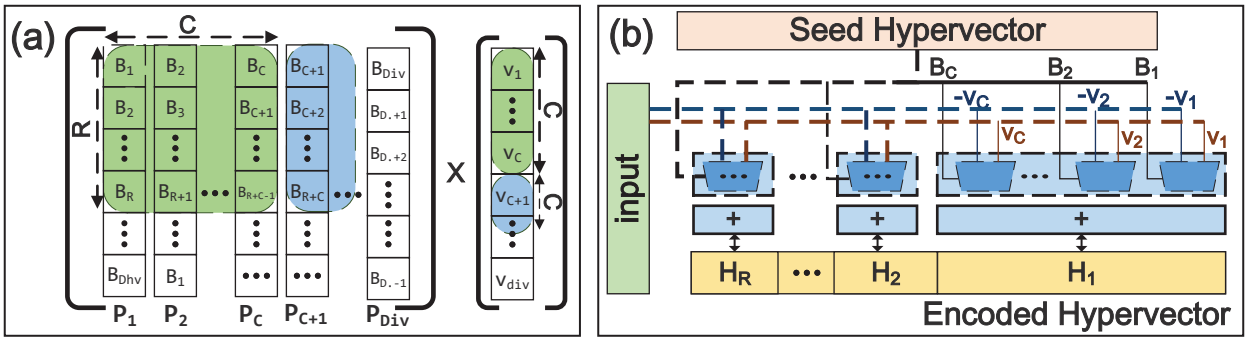

Fig. 4: (a) The matrix multiplication of the projection HVs and the input feature vector. (b) HD2FPGA architecture based on random-projection encoding

the class HVs. It multiplies the generated dimensions of the encoded HV by the corresponding elements of the class HVs and accumulates the multiplied results. After processing all the dimensions of the encoded HV, the class with the highest similarity metric is the prediction result. During the inference step, the predicted label is written into the FPGA off-chip memory.

HD2FPGA classification and clustering kernels share the same EncodeUnit and InputStream modules. The only difference is in the SearchUnit, as illustrated in Figure 5. In retraining the classification model, the HD kernel performs the associative search and finds the most similar class to the encoded HV and compares the predicted label with the actual label, and in case of misprediction, it adds the encoded HV to the class HV of the actual label and subtracts the encoded HV from the mispredicted class. However, in training the clustering HD model, we do not use the actual labels. The HD clustering kernel initializes each centroid HV with the encoded HV of a randomly selected input. Thereafter, it calculates the similarity between each encoded HV and all the centroid HVs and finds the most similar cluster (predicted centroid). The clustering SearchUnit also keeps a copy of the clustering HD model (temp Centroids), and it adds the encoded HVs to the predicted centroid of the temporary model. At the end of each retraining iteration, the SearchUnit replaces the HD clustering model with the temporary model. HD2FPGA clustering kernel, similar to the classification kernel, encodes the input in the first iteration of clustering and stores the encoded HVs into the FPGA off-chip DRAM. In the next iterations, it only reads the encoded HVs from FPGA DRAM to avoid the costly encoding step.

B. HD2FPGA *software*

As shown in Figure 2, the execution of HD2FPGA is split between a host program and the HD kernel with a communication channel between them. The host program, written in C++ and using OpenCL APIs, runs on the host processor, while the HD kernel runs on the FPGA. The API calls, managed by XRT, are used to process transactions between the host program and the HD kernel through the PCIe bus between the host CPU and the FPGA. These communications include transferring the control signals to/from the HD kernel as well as transferring the dataset from the host CPU to the FPGA global memory (DRAM). The host memory is only accessible to the host CPU while the FPGA global memory is accessible by both the host processor and FPGA kernels, therefore, the host is responsible for transferring the dataset to the FPGA DRAM and reading the results from the FPGA DRAM upon kernel completion.

Since the host CPU is responsible for orchestrating the data transfer and kernel initiation, HD2FPGA has no limitation in the size of the dataset. For instance, if the dataset size is greater than the available FPGA's off-chip memory, the host splits the data into the chunks that fit into the FPGA DRAM, sends the commands to the FPGA to process the chunk of the data, and then transfers the other chunks. This is possible since HD training, retraining, and inference are independent of the other chunks. Execution of HD2FPGA host program can be divided into three steps:

Data setup: the host program reads the dataset in the compressed format, quantizes the features if necessary, allocates the corresponding space in the FPGA global memory, and transfers the data to the FPGA global memory through the PCIe bus. A user can also load an already trained HD model, for which the host program can read the trained class HVs from a file specified by the user and execute the inference with the loaded classes.

HD kernel setup: The host program sets up the kernel with its input parameters as well as pointers to the data in the FPGA memory. The input parameters to the HD kernel are the number of data inputs, the length of the HVs, and the HD task (training, retraining, and inference).

HD execution: The host program triggers the execution of the HD kernel on the FPGA. The HD kernel performs the required computations while reading the input data from the FPGA memory and writes the prediction results and/or the trained model back to the FPGA memory.

HD results: The HD kernel, upon compilation, notifies the

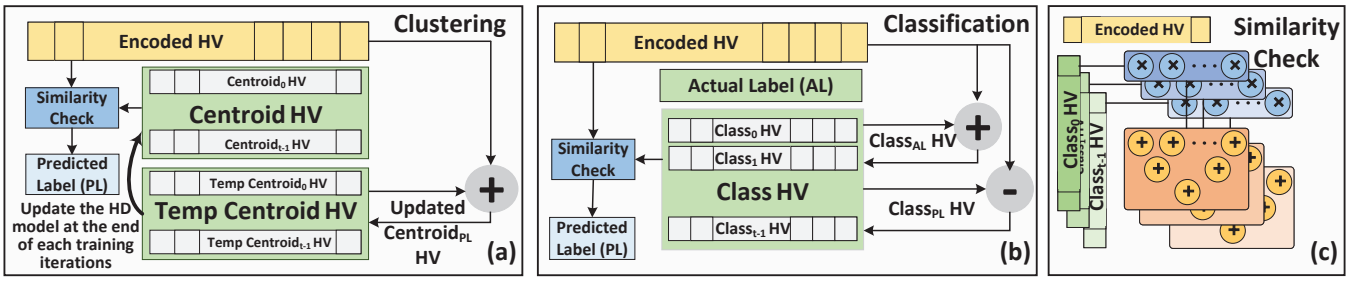

Fig. 5: The proposed architecture of the Search unit for (a) HD Clustering, (b) HD Classification and (c) Similarity Check.

host that it has completed the task. The host program measures the execution time and reads the resulting data (e.g. prediction results and/or trained class HVs) back from FPGA global memory into the host memory.

IV. Experimental Results

HD2FPGA is a flexible framework for efficient implementation of different HD computing applications in FPGA hardware, based on application specifications and user requirements. HD2FPGA is equipped with a GUI written in `Python` which gets the input parameters from the user and generates the header files for both the host program and the HD kernel. The host program has been implemented in `OpenCL` and executes on the CPU. HD2FPGA GUI also synthesizes and generates the FPGA bitstream for the HD kernel of the user application. The HD kernel is written in `C++` and optimized for high performance. HD kernel is synthesized using the Vivado High-Level Synthesis tool (HLS) and integrated with the host code using Xilinx Vitis Accel 2019.2 and is running on U280 FPGA board. To measure the performance of the end-to-end execution of HD classification and clustering on FPGA, we used OpenCL event profiling. We report end-to-end execution times, including reading the data from the FPGA DRAM, executing HD on the FPGA, and writing the results back to the FPGA DRAM. We measure the FPGA board power using the `xbutil query` utility. The energy consumption is computed by multiplying the power obtained using the above method and the FPGA kernel time.

We compare the performance and energy efficiency of HD2FPGA running on FPGA with Intel i7 7600 CPU with 16GB memory. Additionally, we compare the performance of HD classification with F5HD [5] which is the state-of-the-art FPGA-based implementation. We did not compare the energy efficiency of HD2FPGA to F5HD [5] since it does not consider the DRAM power consumption, while in HD2FPGA we measure the exact power consumption of the FPGA board including DRAM and other peripherals.

We evaluate HD on a wide range of benchmark datasets. The HD model is implemented in `PyTorch` to get the accuracy. These datasets cover a broad spectrum of signal classification tasks commonly encountered in edge-sensing applications ranging from human activity detection to text recognition. The publicly available datasets we use are summarized in table I.

The accuracy results are also shown in table I. The HD2FPGA adopts $D_{hv} = 2K$ while F5-HD adopts $D_{hv} = 10K$. Since F5HD utilizes a simplified encoding method, it has to use higher dimensionality for HVs $D_{hv} = 10K$ to achieve comparable accuracy. F5HD with $D_{hv} = 10K$ is more accurate than HD2FPGA with $D_{hv} = 2K$ for the EMG dataset. For clustering, we use the adjusted mutual information score for accuracy. We observe that $D_{hv} = 2K$ is sufficient to achieve 1.0 on some datasets.

A. HD Classification

End-to-End Training: In Figure 6, we combine the encoding, initial training step, and the retraining times of CPU and FPGA to obtain the total training time (including the encoding and 50 retraining epochs). The complexity is proportional to the number of input features, the length of HVs (D_{hv}) and the size of the dataset (number of samples). We show results for $D_{hv} = 2K$ and $D_{hv} = 10K$ with $C = 32$ and two different parallelization levels ($R = 32$ and $R = 256$). Accordingly, we observe that HD2FPGA provides $1.8\times$ (with $R = 32$) and $12.9\times$ (with $R = 256$) speed-up for HVs with $D_{hv} = 2K$ dimensions. For $D_{hv} = 10K$, we achieve $4.2\times$ ($R = 32$) and $30.5\times$ ($R = 256$) speed-up compared to the CPU baseline. By moving from $D_{hv} = 2K$ to $D_{hv} = 10K$, the relative efficiency of FPGA in encoding increases but it decreases for retraining. However, since the encoding time dominates the retraining time of the CPU, in $D_{hv} = 10K$ HD2FPGA achieves higher improvement for the training time. Compared to state-of-the-art classification accelerator, F5HD [5], HD2FPGA with $D_{hv} = 2K$ and $R = 256$ outperforms F5HD by $4.2\times$ while delivering better or equal accuracy. In applications, where the bottleneck is the associative search step, such as ISOLET, HD2FPGA shows higher performance improvement (up to $8.2\times$) compared to F5HD.

To evaluate the energy efficiency of HD2FPGA compared to the CPU baseline, we measure the power consumption of HD2FPGA and CPU during the runtime. Regardless of the benchmark, we observe a power of $\sim 65W$ for CPU encoding, and $\sim 40W$ for its retraining. Figure 7 shows the training energy for CPU and FPGA, comprising the sum of encoding and 50 epochs of retraining energy. HD2FPGA achieves $4.3\times$ ($R = 32$) and $25\times$ ($R = 256$) training energy efficiency for $D_{hv} = 2K$. With $D_{hv} = 10K$, the energy efficiency increases to $12.8\times$ ($R = 32$) and $73.9\times$ ($R = 256$). In the ISOLET dataset, HD2FPGA consumes $\sim 20W$ for $R = 32$ configuration, and $\sim 25W$ for $R = 256$. The difference between the power consumption of these two configurations is

TABLE I: Dataset Information

Dataset	Type	# Classes	# Train	# Test	# Features	HD2FPGA Accuracy	F5-HD[5] Accuracy
CARDIO [16]	**Classification**	3	1,913	213	21	84.5%	84.3%
EMG5 [17]	**Classification**	5	1,473	490	4	72.3%	88.7%
FACE [18]	**Classification**	2	22,441	2,494	608	94.9%	95.6%
UCIHAR [19]	**Classification**	6	6,213	1,554	561	93.7%	93.6%
ISOLET [20]	**Classification**	26	6,238	1,559	617	93.5%	94.2%
Hepta [21]	**Clustering**	7	N/A	212	3	1.0	N/A
Tetra [21]	**Clustering**	4	N/A	770	3	1.0	N/A
TwoDiamonds [21]	**Clustering**	2	N/A	800	2	0.820	N/A
Wingnut [21]	**Clustering**	2	N/A	1016	2	0.909	N/A
Iris [21]	**Clustering**	3	N/A	150	3	0.915	N/A

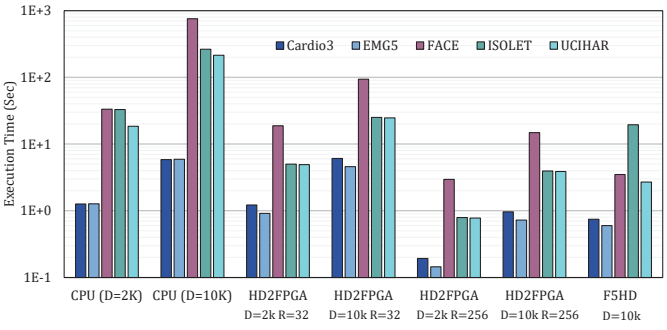

Fig. 6: Training (encoding + 50 epochs retraining) time.

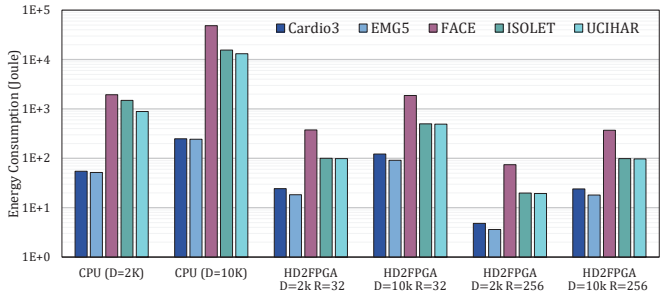

Fig. 7: Training (encoding + 50 epochs retraining) energy consumption.

not proportional to their resource utilization due to the power consumption of the U280 platform.

Inference: Figure 8 compares the performance of HD inference in HD2FPGA compared to CPU. The inference includes the encoding and search steps for HD classification. The SearchUnit module of HD2FPGA is fully pipelined with The EncodeUnit module, meaning that in HD2FPGA the execution time is the maximum of the execution time of the encoding and associative search as they are executing simultaneously in a pipeline fashion. Depending on the application parameters (number of features per input and number of classes), either the encoding or associative search steps can become bottleneck. In Cardio and EMG the search module is the bottleneck, as they have a relatively low number of input features and consequently less complex encoding. In FACE, ISOLET, and UCIHAR, the encoding is computationally more complex than the associative search. For $D_{hv} = 2K$, HD2FPGA achieves $19.5\times$ (with $R = 32$) and $140\times$ ($R = 256$) speed-up over the CPU baseline. For $D_{hv} = 10K$, HD2FPGA inference is $80.0\times$ faster when $R = 32$ and the speed-up further increases to $578\times$ when the matrix-vector multiplication is configured to use 256 parallel rows ($R = 256$). Compared to state-of-the-art F5HD classifier [5], HD2FPGA with $D_{hv} = 2K$ and $R = 256$ provides $1.5\times$ speedup with higher classification accuracy. If the associative search step is the bottleneck, similar to training results, HD2FPGA shows higher performance improvement compared to F5HD [5]. In ISOLET, HD2FPGA provides $2.5\times$ higher performance.

Figure 9 compares the energy consumption of HD2FPGA compared to the CPU baseline. In inference, the encoding step is usually the computational bottleneck and hence, it dominates energy consumption. As HD2FPGA significantly outperformed the CPU baseline in encoding, we expect higher energy efficiency for inference. According to Figure 9, HD2FPGA is $63.3\times$ ($R = 32$) and $366\times$ ($R = 256$) more energy efficient than the CPU baseline for $D_{hv} = 2K$. When $D_{hv} = 10K$ is used, the energy reduction increases to $260\times$ and $1503\times$ respectively.

Compared to the GPU-based OpenHD running on the NVIDIA Jetson TX2 [7], HD2FPGA is $36.6(= \frac{1172us}{32us})\times$ faster and has $5.4(= \frac{4377uJ}{810uJ})\times$ higher energy efficiency in ISOLET, the largest dataset reported in [7], with comparable accuracy(93.5% vs 93.7%[7]).

B. HD Clustering

HD clustering algorithm involves encoding the input data, selecting a subset of the encoded HVs as centroids, assigning the similar encoded HVs to the same centroids, and bundling the assigned HVs to create new centroids. It is similar to the HD encoding and retraining algorithms, where the retraining step finds a similar class (centroid) and in case of misprediction, performs vector-wise addition. The accuracy

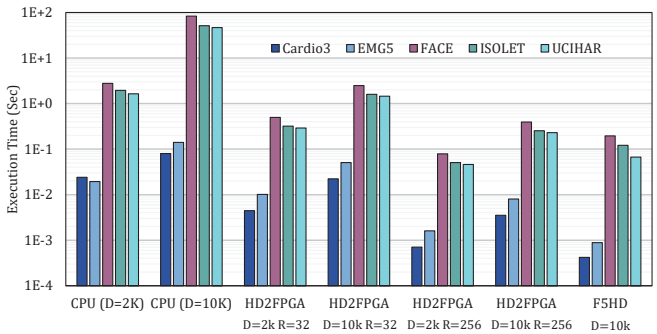

Fig. 8: Inference (encoding + associative search) time.

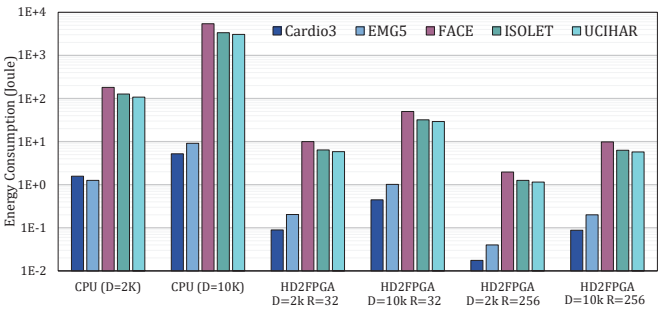

Fig. 9: Inference (encoding + associative search) energy consumption.

(normalized mutual info score) result is shown in Table I. The HD2FPGA adopts $D_{hv} = 2K$. Figure 10 shows the execution time for one epoch of clustering. Since clustering repeatedly performs encoding and search, the execution time linearly increases with the number of training epochs. In all datasets but ISOLET, the search step is the bottleneck of the FPGA pipeline as the datasets contain a small number of features per input, making the encoding comparatively faster than search among the centroids. ISOLET has a significantly higher number of input features and thus, the encoding step in ISOLET is more complicated than the other datasets. With $D_{hv} = 2K$, HD2FPGA performance outperforms the CPU by $3.5\times$ for $R = 32$, and $22.1\times$ for $R = 256$. With $D_{hv} = 10K$, HD2FPGA is $9\times$ faster when $R = 32$, and $57\times$ faster when $R = 256$. Compared to openHD [7], HD2FPGA ($R = 32, D_{hv} = 2K$) is $2.2(= \frac{249ms}{112ms})\times$ faster in TwoDiamonds. Figure 11 shows the energy consumption for the CPU baseline and HD2FPGA clustering. The encoding and search steps consume the same amount of power in the classification step as the algorithm (hence the architecture) is the same. With 2K dimensionality, the HD2FPGA has $10.2\times$ and $51.6\times$ higher energy efficiency for $R = 32$ and $R = 256$, respectively. For $D_{hv} = 10K$, HD2FPGA energy efficiency increases to $27.7\times$ (for $R = 32$) and $140\times$ (for $R = 256$). As mentioned earlier, the execution time of HD2FPGA increases linearly with D_{hv}, while in CPU we observe a larger increase (e.g., the ISOLET encoding runtime increases by $29.7\times$ when moving from $D_{hv} = 2K$ to $D_{hv} = 10K$). Thus, with larger dimensions, HD2FPGA 's energy efficiency further increases.

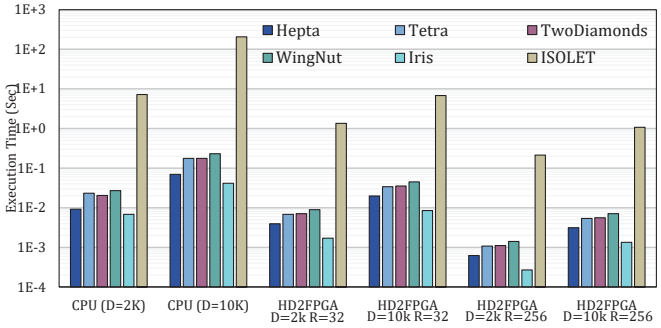

Fig. 10: Clustering (encoding + associative search) time.

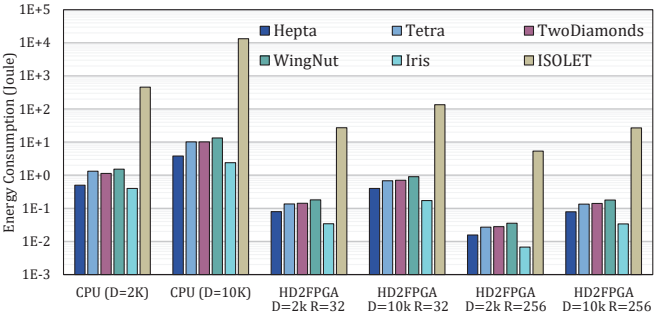

Fig. 11: Clustering (encoding + associative search) energy consumption.

V. Conclusion

In this paper, we propose HD2FPGA, an automated framework for FPGA-based acceleration of HD classification and clustering. HD2FPGA abstracts away the complexities of designing hardware accelerators from an end user. Now it depends on Xilinx's Vivado HLS, but it will be easy to extend to other HLS tools by only changing the pragma to similar ones used in the specific tool. The proposed framework enables a user to specify HD application parameters (e.g., the number of input features, classes, and training data) as well as the application task (classification or clustering) and it accordingly generates a customized FPGA implementation. HD2FPGA supports end-to-end training and inference of HD classification and end-to-end execution of HD clustering on FPGAs. For HD classification, HD2FPGA provides $1.5\times$ ($36.6\times$) speedup compared to state-of-the-art FPGA-based (GPU-based) accelerator framework while providing the same accuracy. For HD clustering, HD2FPGA provides $2.2\times$ speed up compared to the GPU-based accelerator. HD2FPGA can also easily support other applications. For example, regression can be implemented by only modifying the retraining stage by multiplying the error between the actual output and predicted result and learning rate query HV to update the class HV [22].

Acknowledgment

This work was supported in part by PRISM, CoCoSys and CRISP, centers sponsored by SRC and DARPA JUMP 1.0 and 2.0 programs, SRC Global Research Collaboration (GRC) grants, and NSF grants #1826967, #1911095, #2003277, #2003279, #2112665, #2112167, and #2100237.

REFERENCES

[1] 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.

[2] 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.

[3] S. Salamat, M. Imani, and T. Rosing, "Accelerating hyperdimensional computing on fpgas by exploiting computational reuse," *IEEE Transactions on Computers*, vol. 69, no. 8, pp. 1159–1171, 2020.

[4] B. Khaleghi, S. Salamat, A. Thomas, F. Asgarinejad, Y. Kim, and T. Rosing, "Shear er: highly-efficient hyperdimensional computing by software-hardware enabled multifold approximation," in *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design*, pp. 241–246, 2020.

[5] S. Salamat, M. Imani, B. Khaleghi, and T. Rosing, "F5-hd: Fast flexible fpga-based framework for refreshing hyperdimensional computing," in *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 53–62, 2019.

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

[7] J. Kang, B. Khaleghi, T. Rosing, and Y. Kim, "Openhd: A gpu-powered framework for hyperdimensional computing," *IEEE Transactions on Computers*, 2022.

[8] 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.

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

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

[11] 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.

[12] L. Ge and K. K. Parhi, "Classification using hyperdimensional computing: A review," *IEEE Circuits and Systems Magazine*, vol. 20, no. 2, pp. 30–47, 2020.

[13] Y. Guo, M. Imani, J. Kang, S. Salamat, J. Morris, B. Aksanli, Y. Kim, and T. Rosing, "Hyperrec: Efficient recommender systems with hyperdimensional computing," in *2021 26th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 384–389, IEEE, 2021.

[14] A. Menon, A. Natarajan, R. Agashe, D. Sun, M. Aristio, H. Liew, Y. S. Shao, and J. M. Rabaey, "Efficient emotion recognition using hyperdimensional computing with combinatorial channel encoding and cellular automata," *arXiv preprint arXiv:2104.02804*, 2021.

[15] G. Karunaratne, M. Le Gallo, G. Cherubini, L. Benini, A. Rahimi, and A. Sebastian, "In-memory hyperdimensional computing," *Nature Electronics*, vol. 3, no. 6, pp. 327–337, 2020.

[16] "Uci machine learning repository." https://archive.ics.uci.edu/ml/datasets/cardiotocography.

[17] "Uci machine learning repository." https://archive.ics.uci.edu/ml/datasets/EMG+data+for+gestures.

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

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

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

[21] A. Ultsch, "U* c: Self-organized clustering with emergent feature maps.," in *LWA*, pp. 240–244, Citeseer, 2005.

[22] A. Hern?ndez-Cano, C. Zhuo, X. Yin, and M. Imani, "Reghd: Robust and efficient regression in hyper-dimensional learning system," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*, pp. 7–12, 2021.