F5-HD: Fast Flexible FPGA-based Framework for Refreshing Hyperdimensional Computing

Sahand Salamat, Mohsen Imani, Behnam Khaleghi, and Tajana Rosing
Computer Science and Engineering Department, UC San Diego, La Jolla, CA 92093, USA
{sasalama, moimani, bkhaleghi, tajana}@ucsd.edu

ABSTRACT

Hyperdimensional (HD) computing is a novel computational paradigm that emulates the brain functionality in performing cognitive tasks. The underlying computation of HD involves a substantial number of element-wise operations (e.g., addition and multiplications) on ultra-wise hypervectors, in the granularities of as small as a single bit, which can be effectively parallelized and pipelined. In addition, though different HD applications might vary in terms of number of input features and output classes (labels), they generally follow the same computation flow. Such characteristics of HD computing inimitably matches with the intrinsic capabilities of FPGAs, making these devices a unique solution for accelerating these applications.

In this paper, we propose **F5**-HD, a fast and flexible FPGA-based framework for **refreshing** the performance of HD computing. F5-HD eliminates the arduous task of handcrafted designing of hardware accelerators by automatically generating an FPGA implementation of HD accelerator leveraging a template of optimized processing elements, according to the applications specification and user's constraint. Our evaluations using different classification benchmarks revealed that F5-HD provides 86.9× and 7.8× (11.9× and 1.7×) higher energy efficiency improvement and faster training (inference) as compared to an optimized implementation of HD on AMD R9 390 GPU, respectively.

CCS CONCEPTS

• **Hardware → Reconfigurable logic and FPGAs**; Electronic design automation; • **Computing methodologies** → *Machine learning approaches.*

KEYWORDS

Brain-inspired Hyperdimensional Computing; Machine Learning; FPGA-based Acceleration; Automated Template-based Hardware Generation

ACM Reference Format:
Sahand Salamat, Mohsen Imani, Behnam Khaleghi, and Tajana Rosing. 2019. F5-HD: Fast Flexible FPGA-based Framework for Refreshing Hyperdimensional Computing. In *The 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA '19), February 24–26, 2019, Seaside, CA, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3289602.3293913

FPGA '19, February 24–26, 2019, Seaside, CA, USA

ACM ISBN 978-1-4503-6137-8/19/02…$15.00
https://doi.org/10.1145/3289602.3293913

1 INTRODUCTION

Hyperdimensional (HD) computing is a novel computational approach that builds upon imitating the brain functionality in performing cognitive tasks [1, 2]. In fact, brain computes with patterns of neural activity, which can be realized by points in a hyperdimensional space, called hypervectors. By leveraging a non-complex and parallel set of operations on such ultra-wide vectors, HD affords promising capabilities in learning and classification applications including but not limited to language, speech, activity, and face recognition as well as classification of time-series signals [3–9]. In addition to its inclusive cognitive application space and comparatively simpler computation model than other learning paradigms[10, 11], HD computing is inherently robust against failures as the information in a hypervector is uniformly distributed over all of its comprising dimensions [1]. Moreover, HD is able to yield the accuracy of state-of-the-art while learning from only a small portion of the original training data [12, 13].

In a nutshell, HD computing is involved with constituting of and processing on hypervectors, wherein a hypervector comprises thousands of bits. For training, first, it generates a fixed set of orthogonal hypervectors each of which represents a specific feature level. Afterward, for a given input (as a preprocessed set/vector of features), it maps each feature of the input vector to the corresponding predetermined hypervector. Eventually, all the hypervectors are aggregated, which is basically performed by adding them up [3, 14]. Since the spatial or temporal location of the features does matter, the aggregation also incorporates shift operation on the representing vectors to retain the indices of the input features. After all input data are mapped to a final encoded hypervector, all encoded hypervectors belonging to the same class (label) are summed up to form the final representative hypervector of the class. Inference in HD computing is analogous; albeit the encoded hypervector passes through an associative search (a.k.a similarity check) with the representative hypervectors to identify the associated class [1].

The encoding and classifying stages of HD computing require a substantial number of bit-level addition and multiplication operations, which can be effectively parallelized [13]. These operations can also be segregated (and hence, pipelined) in the granularity of dimension level. Though they may vary in the number of input features and output classes, all HD applications follow the same computation flow, albeit with a controllable degree of parallelism and pipeline. Such characteristics of HD computing inimitably matches with the intrinsic capabilities of FPGAs [15], making these devices a unique solution for accelerating these applications; however, implementing applications on FPGAs is a time consuming process[10, 16].

In this paper, we propose F5-HD, an automated FPGA-based framework for accelerating HD computing that abstracts away the implementation complexities and long design cycles associated with hardware design from the user. F5-HD generates a synthesizable Verilog implementation of HD accelerator while taking the high-level user and target FPGA parameters into account. Essentially, F5-HD customizes upon a hand-optimized, fully-pipelined template

processing element that can be parallelized according to the user-specified constraints (viz., accuracy and power). F5-HD supports both training and inference as well as model refinement through online, simultaneous, training and inference, so the model can be calibrated without interrupting the normal operation of the system. Specifically, this paper makes the following contributions:

- Proposes F5-HD, a template-based framework that generates FPGA-based synthesizable architectures for accelerating HD computing.
- Proposes a novel hardware-friendly encoding approach that reduces the required Block RAM accesses, hence, enhances resource utilization
- Provides the flexibility of customized accuracy by supporting different data-types (viz., fixed-point, binary, and power-of-two), and of customized power consumption bound by trading the parallelism.
- Enables simultaneous training and inference to refine the model without interrupting the system functionality.

Our evaluations using different classification benchmarks revealed that, in high-accuracy mode, F5-HD can provide 86.9× and 7.8× (11.9× and 1.7×) higher energy efficiency improvement and faster training (inference) as compared to an optimized implementation of HD on AMD R9 390 GPU, respectively. In the fastest mode in which each dimension is represented by a single bit (i.e., binary), F5-HD achieves 4.3× higher throughput and 2.1× throughput/Watt as compared to the baseline F5-HD using fixed-point values, while providing in average 16.5% lower classification accuracy. In addition, we observe that F5-HD framework can ensure the power consumption to be within 9.0% of the user-defined constraint, on average.

2 BACKGROUND AND RELATED WORK

In this section, we first articulate the operations behind HD computing, including encoding, training, inference, and retraining. Afterward, we review the previous work regarding the utilization and implementation of the HD computing.

2.1 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 hypervectors [1]. In other words, brain computes with patterns of neural activity, which can be better represented by hypervectors rather than scalar numbers. A hypervector 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 hypervectors 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 [3, 14, 17].

Encoding: As demonstrated in Figure 1, training an HD model involves a three-step procedure as follows. First, it initializes base hypervectors, each of which corresponds to a specific input feature level. Indeed, input of the HD algorithm is a feature vector $\vec{V}_{iv}$ with $\mathcal{D}_{iv}$ dimensions (elements) wherein each dimension represents a feature value $\mathcal{F}$ that has ℓ_{iv} levels:

$$\begin{aligned} \vec{V}_{iv} &= \langle v_0, v_1, \cdots, v_{\mathcal{D}_{iv}} \rangle \\ |v_i| &\in (\mathcal{F}_0, \mathcal{F}_1, \cdots \mathcal{F}_{\ell_{iv}}) \end{aligned} \tag{1}$$

Though it is application-dependent, typical values for $\mathcal{D}_{iv}$ and ℓ_{iv} might be, respectively, 100s and four–eight for which ℓ_{iv} can be

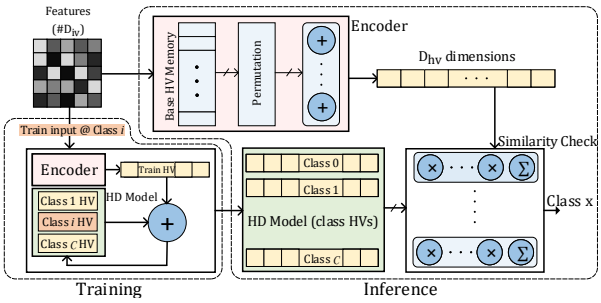


Figure 1: Overview of hyperdimensional learning and inference.

represented by two–three bits. Each of $\mathcal{D}_{iv}$ features in the feature vector needs to be mapped to a base hypervector with $\mathcal{D}_{hv}$ dimensions for subsequent processing. Therefore, to represent all possible ℓ_{iv} values of features, ℓ_{iv} different hypervectors with $\mathcal{D}_{hv}$ dimensions, namely base hypervectors, are needed. The base hypervectors are generated according to the attribute of the feature vector. In the cases that feature levels are independent and irrelevant, base hypervectors can be selected randomly, hence orthogonal. In such cases, the expected Hamming distance between two (out of ℓ_{iv}) base hypervectors is $\sim {}^{\mathcal{D}_{hv}}/_{2}$. However, for the cases that each feature level is a meaningful quantity, e.g., a continuous signal quantized to ℓ_{iv} levels, the distance between the hypervectors of two feature levels should correspond to their actual difference. For these cases, the base hypervector associated with the lowest feature level is generated randomly. Afterward, a random half (${}^{\mathcal{D}_{hv}}/_{2}$) of its bits are flipped to produce an orthogonal base hypervector representing the other side of the horizon, i.e., the highest level of a feature. The remaining base hypervectors are generated by flipping $\frac{{}^{\mathcal{D}_{hv}}/_{2}}{\ell_{iv}-1}$ of each consecutive hypervector pair, starting from the initial base hypervector.

After specifying the base hypervectors, each element v_i of a given input feature vector is mapped to its associated base hypervector $\vec{hv}_{v_i}$ for subsequent processing. Nonetheless, as in most applications the spatial and/or temporal position of an input feature often do matter, i.e., whenever a sequence of the input features should be traced such as image and speech inputs, the encoding procedure takes the locality into account by introducing permutation operation $\mathcal{P}^{(i)}$ (which denotes i-bits cyclic left shift) on the input features before aggregation. Due to the large dimension and randomness of the base hypervectors, $\mathcal{P}^{(i)}$ keeps a hypervector and its resultant shift orthogonal. Eventually, the mapped hypervectors are aggregated according to Equation 2 to build the *query hypervector*:

$$\vec{hv}(\vec{V}_{iv}) = \vec{hv}_{v_0} + (\vec{hv}_{v_1} \ll 1) + \cdots + (\vec{hv}_{v_{\mathcal{D}_{iv}}} \ll \mathcal{D}_{iv}) \tag{2}$$

Which can be reformulated as:

$$\vec{\mathcal{H}} = \vec{hv}(\vec{V}_{iv}) = \sum_{i=0}^{\mathcal{D}_{iv}} \mathcal{P}^{(i)}(\vec{hv}_{v_i}) \tag{3}$$

Training: After mapping each training input $\vec{V}_{iv}$ to hypervector $\vec{\mathcal{H}}$ as above, all hypervectors belonging to the same class (label) are simply summed to form the final representative hypervectors. Thus, assuming $\vec{\mathcal{H}}^l = \langle h_0, h_1, \cdots, h_{\mathcal{D}_{hv}} \rangle^l$ denotes a generated class hypervector for an input data with label l, the final (representative) class hypervectors are obtained as Equation 4, in which each dimension c_k is obtained through dimension-wise addition of

all h_k^ls, and $\mathcal{J}$ is the number of input data with label l.

$$\vec{C}_l = \langle c_0, c_1, \cdots, c_{\mathcal{D}_{hv}} \rangle = \sum_{j=0}^{\mathcal{J}} \mathcal{H}_j^l \tag{4}$$

All dimensions of a class hypervector ($\vec{C}$) have the same bit-width which can have various representation, e.g., binary (hence one bit), power-of-two (2^n), fixed-point (integer), etc. This makes a trade-off between accuracy, performance, and hardware complexity. The base of hypervectors are converted through thresholding. For instance, for $\mathcal{J}$ hypervectors $\vec{\mathcal{H}}_j^l$ constituting class $\vec{C}_l$, the binarized class can be obtained as follows.

$$\vec{C_l}' = \langle c'_0, c'_1, \cdots, c'_{\mathcal{D}_{hv}} \rangle,\ c'_k = \begin{cases} 0 & c_k < \frac{\mathcal{J}}{2} \\ 1 & \text{otherwise} \end{cases} \tag{5}$$

Inference: The first steps of inference in HD computing is similar to training; an input feature vector is encoded to $\mathcal{D}_{hv}$−dimension query hypervector $\vec{\mathcal{H}}$ following Equation 3. This is followed by a similarity check between the query hypervector $\vec{\mathcal{H}}$ and all representative class hypervectors, $\vec{C}_l$. The similarity in the fixed-point and power-of-two number representations is defined as calculating the cosine similarity, which is obtained by multiplying each dimension in the query vector to the corresponding dimension of the class hypervectors, and adding up the partial products:

$$\text{similarity}(\vec{\mathcal{H}}, \vec{C}_l) = \sum_{j=0}^{\mathcal{D}_{hv}} h_k \cdot c_k \tag{6}$$

The class with the highest similarity with the query hypervector indicates the classification result. The number of classes is application-dependent and determined by the user. This can be as simple as two classes, denoting face vs. non-face in a face-detection algorithm. Similarity checking in binarized HD model (i.e., 1-bit dimensions) simplifies to the Hamming distance between the query and class vectors, which can be carried out by a bitwise XNOR, followed by a reduction (population counter[1]) operation.

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 hypervectors from the mispredicted class and adding it to the right class. Thus, for a new input feature vector $\vec{V}_{in}$ with query hypervector $\vec{\mathcal{H}}$ belonging actually to class with hypervector $\vec{C}_l$, if the current model predicts the class $C_{l'}$ where $C_{l'} \neq C_l$, the model updates itself as follows:

$$\begin{aligned} \vec{C}_l &= \vec{C}_l + \vec{\mathcal{H}} \\ \vec{C}_{l'} &= \vec{C}_{l'} - \vec{\mathcal{H}} \end{aligned} \tag{7}$$

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

2.2 Related Studies

HD computing is gaining traction as an alternative solution to perform cognitive tasks in a light-weight fashion that uses significantly simpler operations compared to conventional machine learning techniques that deal with complex learning procedures with substantial number of costly operations. So far, successful application of HD computing in varied domains has been demonstrated. language identification [18], DNA sequencing [19], physical activity prediction [5, 20], speech recognition [6, 21], and gesture recognition [12, 22], clustering [23] are just a few examples.

On par with studies investigating the HD applications, several studies have attempted to propose hardware and algorithmic solutions to enhance the efficacy of HD computing. The study in [17] proposes logical operations to generate the hypervector corresponding to each feature on the fly, in order to reduce the costly BRAM accesses. They also propose approximate majority gate to compose the binary class hypervectors without requiring to hold the summation on hypervector components in a multi-bit format in the course of training. This is, however, limited to low-accuracy binarized HD computing wherein each dimension of the query and class hypervectors is one bit. The authors of [13] propose hierarchical HD computing solution that consists of a main stage with multiple classifiers each can trade between efficiency and accuracy. There is also a decider stage that learns and selects the appropriate encoder within the main stage based on a so-called difficulty metric of the input data. The work in [21] clusters class hypervectors dimensions to reduce the number of multiplications. Additionally, by assuming the encoded input hypervector is stored in memory, they implemented the associative search of clustered HD on FPGA.

Other works leverage advances of emerging technologies in HD computing [24–26]. In [24], the authors leverage CNT-FET and Resistive RAM to fabricate an end-to-end HD computing solution. They exploit the variations in RRAM resistance and CNT-FET drives current to project the input features to query hypervectors as well as propose approximate accumulation circuit using gradual RRAM reset operation. The work in [25] demonstrates HD computing with 3D vertical RRAM in-memory kernels capable of performing multiplication, addition, and permutation by analog operations on RRAM cells.

To the best of our knowledge, F5-HD is the first automated FPGA-based framework that implements HD computing with varied model precision, capable of meeting user constraints on different FPGA platforms.

3 F5-HD FRAMEWORK OVERVIEW

F5-HD aims to abstract away the complexities behind employing FPGAs for accelerating AI applications [27]. F5-HD is an automated framework that generates synthesizable FPGA-based HD implementation in Verilog, considering the user-specified criteria, e.g., power budget, performance-accuracy trade-off, and FPGA model (available resources). F5-HD combines the advantages of hand-optimized HDL design with the bit-level yet flexible manageability of FPGA resources, which is in concordance with bitwise operations associated with HD computing, to accelerate these applications.

3.1 F5-HD Workflow

Figure 2 demonstrates F5-HD's workflow, explained as follows.

(1) Model Specification: The framework starts with specifying the application specifications, viz., the number of classes, features (i.e., input vector dimensions $\mathcal{D}_{iv}$, as well as the number of features different levels, ℓ_{iv}) and the number of training data. The user also determines the target FPGA model, hence F5-HD can get the number of available resources from a predefined library. F5-HD currently supports Xilinx 7-series FPGAs, including Virtex-7,

[1] A population counter basically counts the number of '1's (ones) in the given binary input.

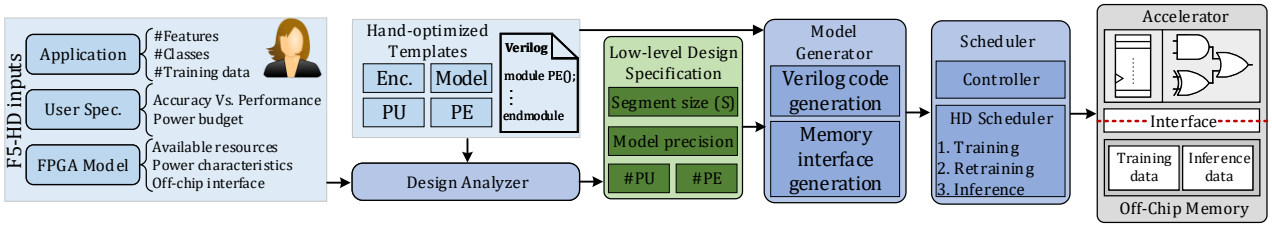


Figure 2: Overview of the proposed framework, F5-HD.

Spartan-7, and Kintex-7 families. This can be readily extended to other FPGA families. In addition, the user can dictate constraints on the power as well as performance-accuracy trading, which will be explained in the following subsections.

(2) Design Analyzer: Thereafter, F5-HD's design analyzer determines the number of resources according to the user's specification. F5-HD exploits a parameterized template architecture, mainly composed of an encoder; an associative search unit, including Processing Units and Processing Elements; as well as an HD model module that stores and updates the class hypervectors. The hardware architecture of F5-HD will be detailed in Section 4. The design analyzer determines the number of *Processing Units* (PUs), *Processing Elements* (PEs) as well as the type and number of dimension-wise functional units within each PE, according to the desired accuracy level and available resources. All the function units, e.g., encoder and PUs, utilize a specific set of building blocks with foreknown resource utilization. Thus, F5-HD design analyzer can readily figure out the parameters of the template architecture, e.g., maximum parallelization level of the encoder (see Section 4.1) and number of PEs per PU, based on their required resources (LUT, BRAM, and DSP) and the available resources.

In the case a power budget is defined by the user, the design analyzer tries to find out the maximum number of PEs that can be generated, without violating the constraints. For this regard, F5-HD estimates the power of resources, e.g., LUTs, flip-flops, DSPs, BRAMs, etc. using Xilinx Power Estimator (XPE) [28]. This requires calculating the expected activity of the resources, which is straightforward owing to the foreknown homogeneous structure of the generated architectures and the expected probability of the hypervectors at the level of the dimension. Another constraint is performance-accuracy trade-off wherein the user chooses between the highest performance with relatively lower accuracy, mediocre, and low performance with the highest accuracy. The available modes are currently fixed-point (8-bits integer representation), power-of-two in which hypervector dimensions are four-bits values that represent the exponent, and binary (i.e., each dimension is represented by one bit). It is noteworthy that the power and accuracy constraints can be applied concurrently, which provides the user with the flexibility to adapt F5-HD based on their application criteria. For instance, for real-time low-power applications, the user might specify their power budget with the binary mode of operation. The output of design analyzer is basically the number of PUs and PEs (per PU), the number of multipliers (in the case of fixed-point model) per PE, and the parallelization level of the encoder, i.e., the number of hypervector dimensions it can produce at each cycle.

(3) Model Generator: After the design analyzer specified the parameters of the template architecture, F5-HD's model generator, automatically generates the Verilog implementation of F5-HD using hand-optimized template blocks. This includes instantiating the PUs, PEs, the Block RAMs, and off-chip memory interface.

Table 1: Classification accuracy and performance of binary, power-of-two, and 8-bits fixed-point HD models running on CPU.

Application	Binary		Power-of-two		Fixed-point	
	Accuracy	Exe.time	Accuracy	Exe.time	Accuracy	Exe.time
Speech Recognition	88.1%	1.6ms	90.3%	3.4ms	95.5%	10.5ms
Activity Recognition	77.4%	0.6ms	88.0%	1.3ms	94.6%	3.4ms
Face Recognition	48.5%	0.7ms	89.6%	1.6ms	96.9%	4.6ms
Physical Monitoring	85.7%	1.1ms	90.8%	2.4ms	94.5%	7.8ms

The model generator also initializes the BRAMs with the base hypervectors. For this end, F5-HD exploits a fixed, predetermined hypervector as the seed vector, and generates the remaining $\ell_{iv} - 1$ hypervectors according to the procedure explained in Section 2.1. In the cases the user already has a trained model (i.e., base and class hypervectors), F5-HD allows direct initializing of these hypervectors.

(4) Scheduler: The next step generates the controller, which statically schedules F5-HD operations. The main scheduling tasks include loading the training or inference data from off-chip memory into local BRAMs, switching between the training, inference, and/or retraining modes. It also generates a controller to allocating and deallocating PUs for retraining, and essentially controlling the enabler of different processing units in the granularity of clock cycle. Eventually, the logic and controller are merged to realize the concrete accelerator architecture.

3.2 Accuracy-Performance Trade-off

The majority of existing HD computing methods use binarized class hypervectors to substitute the costly Cosine similarity operation in inference phase with the simpler Hamming distance operation. Although binary representation increases the throughput, in the majority of classification problems, the accuracy of the binarized HD model is not comparable to that of the HD using fixed-point dimensions [13]. In addition to the fixed-point and binary HD models, we provide power-of-two representation in the class hypervectors which replaces the costly multiplication operations with shift operations in the hardware level. Though power-of-two representation covers discrete values, it supports a larger range of numbers which helps to compensate for the accuracy drop. Table 1 compares the accuracy and execution time of HD models for four different datasets on CPU. Fixed-point model, on average, attains 5.7% and 20.5% higher accuracy compared to, respectively, power-of-two and binary models. The binary model surpasses in terms of the throughput, wherein it yields 6.5× and 2.2× performance improvement over the fixed-point and power-of-two models.

3.3 Training Modes

Similar to the training of Deep Neural Networks (DNNs), training of HD model can be enhanced by iterating over the input data, as described in Section 2.1. Note that, as in the case of DNNs, to avoid overfitting, a learned model does not necessarily predict the correct class for all data of the same training dataset, however, the accuracy can be improved by multiple iterations (equivalent to

multiple *epochs* in the context of deep learning). The first epoch of F5-HD generates all query hypervectors (one per each input data) and aggregates the hypervectors with the same label l as the class hypervector $\vec{C}_l$. We denote this single-epoch learning as **model initialization**. During the subsequent optional epochs (referred to as **retraining**), which either can be specified by the user or F5-HD itself continues until the accuracy improvement diminishes, under the management of the scheduler, F5-HD enhances the model by discarding the attributes of the mispredicted query hypervector $\vec{\mathcal{H}}$ from the mispredicted class hypervector $\vec{C}'_{\mathcal{H}}$, and adding it to the correct class hypervector $\vec{C}_{\mathcal{H}}$. Retraining can be carried out immediately after model initialization, or enabled later by halting the inference phase. The principal difference between the model initialization and retraining is the latter requires prediction (i.e., inference) as well while the former simply performs aggregation. This is supported by F5-HD architecture, which is further described in Section 4.

Depending on the generality of the training data and the HD model, in certain cases, the accuracy of the classifier for real-world data might drop. To resolve this issue, F5-HD provides an **online retraining** solution which can be enabled during the runtime by user. During the online retraining, F5-HD updates the class hypervectors based on a new set of training data in real-time. Thus, F5-HD is capable of conducting model initialization, retraining, inference, and simultaneous retraining-inference (online retraining). In the inference mode, the system works normally and all the resources are assigned to calculate the similarity metric. In the online hybrid retraining mode, the system executes both inference and retraining and allocates a portion of the resources for each task. In this mode, the part of the FPGA that executes the inference task always uses the updated model during the online retraining. Therefore, in each retraining iteration, the model is updated and the inference employs the recently updated class hypervectors for prediction. Upon finishing the online retraining, all FPGA resources will be reallocated back for inference purpose.

3.4 Flow of Data

Inputs of F5-HD are vectors of extracted features, namely feature maps, which are stored in the off-chip memory. The scheduler partially loads the feature maps to the input buffer memory, distributed in FPGA local memory (Block RAMs). The encoding module generates the encoded query hypervectors of the input vector and stores them in the encoding buffer. The generated query hypervectors are then pipelined in a segregated (dimensional-wise) manner, fed to the associative search module to perform parallel similarity check with all class hypervectors, yet in a dimensional-wise manner. This requires to store the partial sums of the dimensions products. The encoding and associative search work in a synchronous manner to avoid logic starvation and maximize the physical resource utilization. Thus, in F5-HD, the encoding module outputs the same number of query hypervector dimensions that the associative search processes per cycle. Since the classification of an input vector takes multiple cycles and utilizes all the FPGA resources, the parallelization is in per-input level. That is, classification operations for a single input are pipelined and parallelized among all FPGA resources, and the subsequent input vector is loaded after the process of the current input accomplishes. Increasing F5-HD's throughput necessitates increasing the degree of parallelism in the associative search, which, in turn, demands reading higher encoded dimension per cycle. Therefore, owing to the high supported degree of parallelism in HD computing, the only performance barriers of F5-HD are the available resources and power budget.

4 F5-HD ARCHITECTURE

In this section, we articulate the contributions of F5-HD in more details. We begin with elaborating the proposed encoding scheme that reduces the number of BRAM accesses. Afterwards, we illustrate the architecture overview and detail the functionality and structure of the building blocks in the course of training and inference. We also formulate the required resources by which the design analyzer specifies the (parametric) number of resources for the model generator.

4.1 Proposed Encoding Scheme

Both training and inference processes in HD computing need to encode the input feature hypervector, $\vec{V}_{in}$, to the query hypervector $\vec{\mathcal{H}}$, using basic permutation and addition on the base hypervectors. As previously shown by Equation 3, each element v_i of the input hypervector, based on its value $|v_i| \in (\mathcal{F}_0, \mathcal{F}_1, \cdots \mathcal{F}_{\ell_{iv}})$, selects the corresponding base hypervector $\vec{hv}_{vi}$ (out of ℓ_{iv} possible base hypervectors), rotated left by i bits, to make up the query $\vec{\mathcal{H}}$. Figure 3(a) illustrates the encoding scheme, in which the constituting bits of each dimension d_i of the query hypervector $\vec{\mathcal{H}}$ are distinguished by the same color. Accordingly, to build up e.g., dimension d_0 (d_1) from $\vec{\mathcal{H}}$, v_0 of the input hypervector chooses among b_0 (b_1) of the base hypervectors, v_1 selects from $b_{\mathcal{D}_{iv}}$ (b_0), v_2 selects from $b_{\mathcal{D}_{iv}-1}$ ($b_{\mathcal{D}_{iv}}$), etc. Recall that the dimensions of hypervectors are 1-bit wide (denoted by b_is in the figure) that aggregate in a dimension-wise scheme and form d_is, which can be in various widths and representations, e.g., fixed-point, binary, and power-of-two.

The naïve encoding scheme abstracted in Figure 3 is, however, both computationally and communicationally intractable: at each cycle it requires $\ell_{iv} \times \mathcal{D}_{hv}$ bits (multiples of 10K) of the base hypervectors to be read from the BRAMs, and $\mathcal{D}_{hv}$ population counters (`PopCounters`), each with input bitwidth of $\mathcal{D}_{iv}$. To resolve this, as the dimensions of the query hypervector $\vec{\mathcal{H}}$ can be calculated independently, we segregate the output query vector $\vec{\mathcal{H}}$ into the segments of $\mathcal{S}$ dimensions whereby at each clock cycle one segment is processed. Thus, processing the entire $\vec{\mathcal{H}}$ takes $\mathcal{D}_{hv}/s$ cycles. This is conceptualized in Figure 3(b), which shows the physical locations of the hypervectors bits required to build up the first $\mathcal{S}$ dimensions of $\vec{\mathcal{H}}$. Accordingly, $\ell_{iv} \times (S + \mathcal{D}_{iv})$ different bits are needed to be read to create the query $\vec{\mathcal{H}}$. Notice that this approach retains the alignments of the bits; for every $S + \mathcal{D}_{iv}$ consecutive bits (per base hypervector) read from the BRAM(s) at each cycle, bits 0 to $\mathcal{D}_{iv}$ are conveyed to 0^{th} PopCounter to form d_0, bits 1 to $\mathcal{D}_{iv} + 1$ form the d_1 via the 1^{st} PopCounter, and so on. Therefore, no logic or routing overhead is associated to align the read data.

Beside segmented processing, we further reduce the number of BRAM accesses by proposing a novel encoding scheme. The proposed encoding, first, permutes the bits of the base hypervectors *locally*, i.e., intra-segment, rather than the entire hypervector. After $\mathcal{S}$ permutations, e.g., after the first $\mathcal{S}$ features (v_is) in the input hypervector, the segments accomplish an entire permutation; hence the base hypervector for the 0^{th} and $'\mathcal{S} + 1'^{\text{th}}$ features essentially become the same. This removes the information associated with local and/or temporal locality of the input features. In such case, we perform inter-segment permutation in which the *segments* are

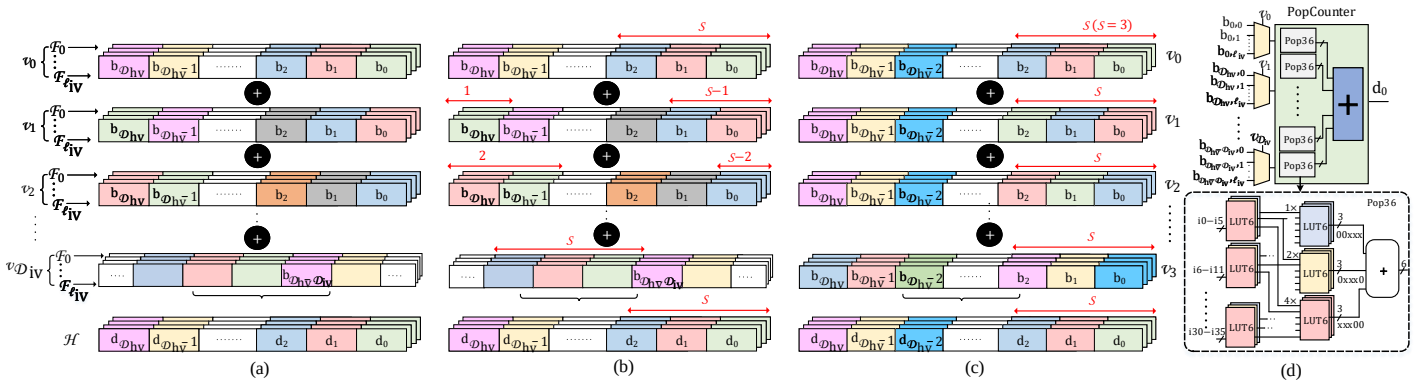


Figure 3: (a) The naïve encoding scheme (b) Baseline segmented encoding (c) The proposed encoding scheme (d) Implementation of the population counter

permuted to left *globally*, whereby bit b_k takes the place of bit $b_{\mathcal{S}+k}$. In this scenario, the first $\mathcal{S}$ features (v_is) need $\mathcal{S}$ bits of the first segment, the second $\mathcal{S}$ input features require $\mathcal{S}$ bits of the right segment (which will be shifted to left by one segment), and so on. Thereby, the proposed encoding needs $\ell_{iv} \times (\mathcal{S} \times {}^{\mathcal{D}_{iv}}/_{\mathcal{S}}) = \ell_{iv} \times \mathcal{D}_{iv}$ bits ($\mathcal{S}$ bits of all ℓ_{iv} base hypervectors per every ${}^{\mathcal{D}_{iv}}/_{\mathcal{S}}$ input features) to produce an output segment. This needs $\mathcal{S}$ $\mathcal{D}_{iv}$-width PopCounter. Figure 3(c) conceptualizes the proposed encoding scheme.

The hand-crafted hardware realization of the proposed PopCounter, which contributes to significant portion of the encoder and overall area footprint, is demonstrated by Figure 3(d). The main building block of the implemented PopCounter is `Pop36` that produces 6-bit output for a given 36-bit input. It is made up of bunches of three LUT6 that share six inputs and output the 3-bit resultants, which are summed up together in the subsequent stage according to their bit order (position). We instantiated FPGA primitive resources, e.g., `LUT6` and `FDSE` to build up the pipelined PopCounter, which is ~20% area efficient than simple HDL description. The impact of PopCounter intensifies further in binary HD models wherein the associative search module is relatively small.

4.2 F5-HD Architecture

The architecture overview of F5-HD is illustrated in Figure 4, which incorporates the required modules for training, inference and online retraining of the HD computing. The main `template` architecture of F5-HD includes two levels of hierarchy: a cluster of *Processing Units* (PUs), each comprises specific number of *Processing Elements* (PEs). The assignment of PUs and PEs are selected in a way that maximizes the data reusability.

Processing Units (PUs): F5-HD contains $2 \times |C|$ PUs where $|C|$ is the number of classes (labels). In the course of inference, all C PUs perform similarity checking. Every cycle, each PU receives ${}^{\mathcal{S}}/_{2}$ of the query hypervector's dimensions (recall that $\mathcal{S}$ is the segment length generated by encoder at each clock cycle, as discussed in Section 4.1). Thus, together, a pair of PUs process all $\mathcal{S}$ dimensions of the segment, and hence, $2 \times |C|$ PUs are able to check similarity between all $|C|$ classes in parallel. Every PU_k also contains a local buffer to prefetch (a portion of) the associated class hypervector C_k in advance to suppress the BRAM's read delay. Additionally, PU includes a pipelined accumulator to sum up and store the results of PEs, to be aggregated with the results of the next ${}^{\mathcal{S}}/_{2}$ dimensions.

Processing Elements (PEs): Each PE contains a predetermined number of multipliers and adders (based on the FPGA size, normally eight fixed-point multipliers). However, the number of PEs in each PU which together with the PopCounters of encoder determine the level of parallelism (value of $\mathcal{S}$), is specified according to the available FPGA resources. The available resources may be restricted by the power budget, as well. PEs generally perform the similarity check through calculating the dot-product of the query and class hypervectors, though it requires different type of operations for different model precision (different representations of dimensions). Typically, PEs consist of fixed-point multipliers, which we map them to FPGA DSPs. Utilizing power-of-two HD model replaces the multiplications with shift operations in which each dimension of the query $\vec{\mathcal{H}}$ is shifted by the value specified by the corresponding element of the class hypervector. Using binary HD model further simplifies this to element-wise XNOR operations, followed by reduction or population count, in F5-HD XNOR and population count operation is combined and implemented in *XS* LUTs followed by a layer of 6-input population count logic (*P6* LUTs). Therefore, the advantage of a hand-crafted PopCounter gets further noticed in the binarized HD models. To generate HD architectures of different accuracy, F5-HD produces PEs with the specific structure, the template architecture is retained.

In the following, we explain how F5-HD architecture splits the processes during the model initialization, inference, and retraining procedures.

Model Initialization: Model initialization starts with randomly initializing of the class hypervectors as well as generating the orthogonal, base hypervectors. Since model initialization is carried out only once in the entire course of the HD computing, we try to simplify this stage and do not allocate specialized resources. Therefore, we load both the base hypervectors and initial (random) class hypervectors during initial programming of the FPGA. Thereafter, all training input data is encoded and then added to the initial class hypervector. We use the same encoding module used for generating the query hypervectors, which, at each cycle, generates $\mathcal{S}$ dimensions of the encoded input vector and adds it back to the corresponding class hypervector using the $\mathcal{S}$-wide adder incorporated in the *model* module (see Figure 4).

Inference: Figure 4 demonstrates the structure of the inference block in F5-HD architecture. The encoded query hypervector $\vec{\mathcal{H}}$ is broadcast to all PUs, each of which shares ${}^{\mathcal{S}}/_{2}$ corresponding dimensions of its prefetched associated class hypervector between its PEs. PUs accumulate the sum-of-the-products to be aggregated with the subsequent segments' results. After processing the entire query hypervector accomplished, i.e., after ${}^{\mathcal{D}_{hv}}/_{\mathcal{S}}$ cycles, the final similarity resultant of each class is obtained by adding the accumulated values of each PU pair. Eventually, the comparator outputs the class index with the greatest similarity metric.

Retraining: Remember from Section 2.1 that during the retraining stage, the HD model performs inference on the same input data

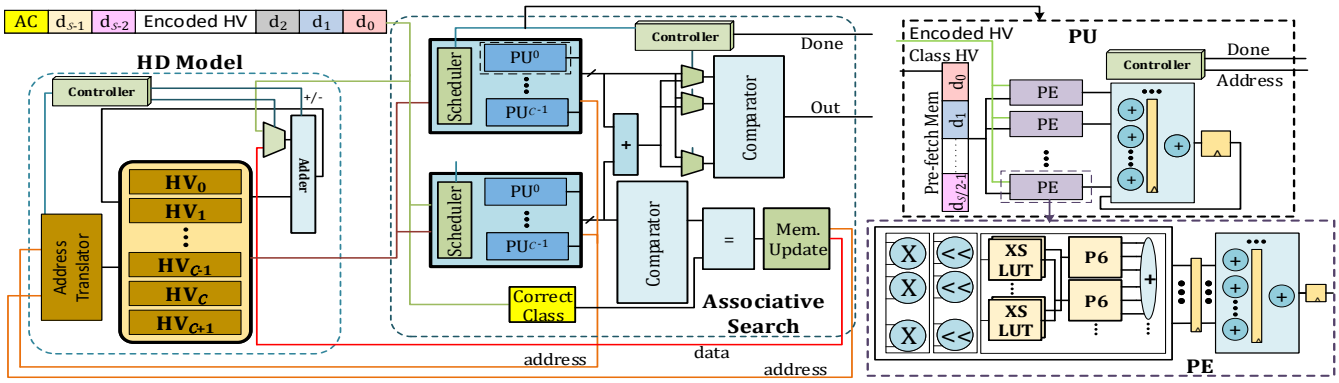


Figure 4: Overview of the HD classification, consisting of HD model, associative search, PUs and PEs structure.

and, in the case of misprediction, updates the necessary classes, i.e., the correct and mispredicted classes. In F5-HD architecture, it is performed by passing the mispredicted query hypervector to the *HD model* module, which adds (subtracts) the query to (from) the correct (mispredicted) class. The correct class index is specified by the label of input data. In summary, retraining involves with inference, followed by a potential model update.

Online Retraining/Inference: In this operating mode, the encoder generates $^{\mathcal{S}}/_{2}$ dimensions for the inference, and $^{\mathcal{S}}/_{2}$ for the retraining data. Using the upper pairs of PUs (see Figure 4), inference executes by $^{1}/_{2}$ of its typical throughput and takes $2\times{}^{\mathcal{D}_{hv}}/_{\mathcal{S}}$ per input. The other half of PUs perform retraining, which, as already discussed, includes an inference followed by a potential model update. In the case of a misprediction which demands a model update, the inference should be halted to update the required classes. To avoid this, we have dedicated two additional hypervectors to write the updated classes (hypervectors). Upon a misprediction, the query hypervector will be subtracted from the mispredicted class, which is already being read by the inference module segment by segment, so no additional read overhead will be imposed. Thereafter, the hypervector will be added to the correct class. After updating each of the correct and mispredicted hypervectors, the address translator modifies the physical address of the two classes to point the right hypervector. Note that till the mispredicted classes are updated, the HD model works with the previous classes.

Resource Constraints: As the number of PUs are fixed, the number and size of PEs (i.e., number of multipliers per PE) per each PU affect the level of parallelism in HD computing. This, however, is also restricted by the number and bandwidth of on-chip RAMs as well as the dictated power budget. The following equations summarize the constraint of different resources F5-HD assumes in generating F5-HD architecture.

$$\overbrace{A_{PopCounter}\times\mathcal{S}}^{encoding}+\overbrace{2\times|C|\times\mathcal{N}_{PE}\times A_{PE}}^{Similairty\ checker}<\text{LUT}_{max} \quad (8)$$

$$\overbrace{2\times|C|\times\mathcal{N}_{PE}\times DSP_{PE}}^{Similairty\ checker}+\overbrace{\mathcal{S}}^{model\ updater}<\text{DSP}_{max} \quad (9)$$

$$\frac{\overbrace{|C|\times S\times bitwidth}^{HD\ model\ read\ access}+\overbrace{\mathcal{D}_{iv}\times\ell_{iv}}^{encoding}}{36}<\text{BRAM}_{max} \quad (10)$$

In these equations, $A_{\mathcal{X}}$ denotes the area of module $\mathcal{X}$ in terms of number of LUTs, $\mathcal{N}_{PE}$ is the number of PEs in each PU, DSP_{PE} is the number of DSPs per PE (in the case of fixed-point models). We also map the adder of the model updater into DSP blocks, as evident from Equation 9. Notice that, in the proposed architecture, the computation is limited by BRAM accesses (rather than BRAM memory). Thus, we have assigned the constraint on BRAM bandwidth. It is also noteworthy that our experiments revealed the design is barely routable for LUT utilization rates above ~90%. Hence, LUT_{max} is set to 90% of the device LUTs.

5 EXPERIMENTAL RESULTS

F5-HD is a flexible framework for efficient implementation of different HD computing applications in FPGA hardware, respecting the application specifications and user's requirements. The entire F5-HD software support including user interface and code generation has been implemented in C++ on CPU. The software customizes template blocks to generate an optimized hardware for each application, based on the user's optimization, accuracy, and power preferences. The output of F5-HD framework is an FPGA-mapped implementation of a given HD application in Verilog HDL. We verify the timing and the functionality of the F5-HD by synthesizing it using Xilinx Vivado Design Suite[29]. The synthesized code has been implemented on Kintex-7 FPGA KC705 Evaluation Kit. We used Vivado XPower tool to estimate the device power.

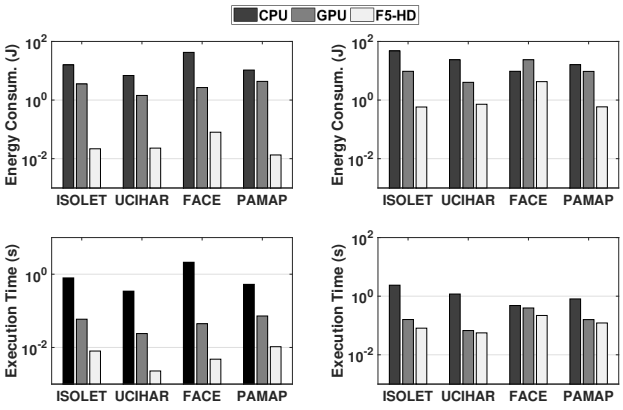
We compare the performance and energy efficiency of F5-HD accelerator running on FPGA with AMD R9 390 GPU and Intel i7 7600 CPU with 16GB memory. For GPU, the HD code is implemented using OpenCL and is optimized for performance. We used Hioki 3334 and AMD CodeXL [30] for the power measurement of CPU and GPU, respectively. We implement F5-HD on three FPGA platforms including Virtex-7 (XC7VX485T), Kintex-7 (XC7k325T), and Spartan-7 (XC7S100) to evaluate the efficacy of F5-HD on various platforms with different available resources, power characteristics and power budget. We evaluate the efficiency of F5-HD on four practical workloads including **Speech Recognition (ISOLET)** [31]: the goal is to recognize voice audio of the 26 letters of the English alphabet, **Activity Recognition (UCIHAR)** [32]: the objective is to recognize human activity based on 3-axial linear acceleration and 3-axial angular velocity, **Physical Activity Monitoring (PAMAP)** [33]: the goal is to recognize 12 different human activities such as lying, walking, etc., and **Face Detection**: the goal is to detect faces among Caltech 10,000 web faces dataset [34] from negative training images, i.e., non-face images which are selected from CIFAR-100 and Pascal VOS 2012 datasets [35].

5.1 Encoding

Encoding module is used in both training and inference. This encoder works in a pipeline stage with the initial training and associative search (similarity checking) modules. Thus, the more generated

Table 2: The maximum number of generated encoded dimensions per cycle using Kintex FPGA

#Features	64	128	256	432	512
Baseline	975	449	254	128	110
F5-HD	1505	837	481	243	211



(a) Training (b) Retraining

Figure 5: Energy consumption and execution time of F5-HD versus other platforms during (a) training and (b) one epoch of retraining.

dimensions by the encoding module, the more throughput F5-HD can achieve. To evaluate the effectiveness of our proposed encoding algorithm, we compare the hardware implementation of F5-HD encoding with a baseline HD computing encoding [13].

Table 2 compares the number of generated dimensions per cycle in F5-HD and the baseline encoding modules. In the baseline segmented encoding, to generate $\mathcal{S}$ dimensions of the encoded hypervector, we showed that HD architecture needs to read $\mathcal{S} + \mathcal{D}_{iv}$ dimensions of each base hypervector, where S and $\mathcal{D}_{iv}$ are the segment length and length of the input hypervector, respectively. In contrast, as we explained in Section 4.1, F5-HD encoding module is implemented using a hardware-friendly permutation as well as LUT-based XNORand PopCount modules that reduces the resource usage. Our evaluation on data points with 64 features shows that F5-HD encoder can provide 1.5× higher throughput as compared to the baseline segmented encoder. This throughput improvement increases to 1.9× for data points with 512 features. This is because the delay of the adder (population counter) dominates as the number of features (hence, the size of the population counter) increases.

5.2 Training

Initial Model Training: HD generates the initial model by a one-time passing through the training dataset. Regardless of the exploited models (viz., binary, power-of-two or fixed-point), in F5-HD we train the HD model using fixed-point operations and eventually we quantize the class hypervectors based on the defined model precision. Figure 5(a) shows the energy consumption and execution time of HD running on Intel i7 CPU, AMD R9 390 GPU, and Kintex-7 FPGA platforms during the initial training. The initial training consists of the encoding module which maps data points into high-dimensional space and hypervectors aggregation which generates a hypervector representing each class. In conventional computing systems, e.g. CPU and GPU, the majority of training time is devoted to the encoding module, since these architectures have not been customized to process binary vectors in 10K dimensions. In contrast, F5-HD can implement the encoding module effectively using FPGA primitives. Our evaluation shows that F5-HD provides, on average, 86.9× and 7.8× (548.3× and 148.2×) higher energy efficiency and faster training as compared to GPU (CPU) platform, respectively.

Retraining: Similarity checking (a.k.a associative search) is the main contributor to HD energy consumption and execution time during both retraining and inference. In retraining, associative search checks the similarity between a fixed-point query hypervector with all stored class hypervectors using cosine metric. Since the HD encoding is expensive on conventional computing units, in CPU and GPU implementations, the retraining processes on the encoded training data which are already stored in memory. In contrast, due to the efficient F5-HD encoding functionality and in order to reduce the off-chip memory access, F5-HD encodes the training data on every iteration. Figure 5(b) compares the HD computing retraining efficiency on three CPU, GPU, and FPGA platforms. The results are reported for F5-HD retraining on a single epoch. Our evaluation shows that F5-HD provides 1.6× and 10.1× faster computation as compared to GPU and CPU platforms, respectively. Although the GPU performance is comparable to F5-HD, F5-HD provides 7.6× higher energy efficiency due to its lower power consumption.

5.3 Inference

Figure 6 compares the energy consumption and execution time of HD inference running on different platforms. All results are reported for the case of using the fixed-point model. The inference includes the encoding and associative search modules. The encoding module maps a test data into high-dimensional space, while the associative search module checks the similarity of the encoded data to pre-stored class hypervectors. The results show that the efficiency of applications changes depending on the number of features and the number of classes. For applications with a large feature size, F5-HD requires a costly encoding module, while applications with a large number of classes, e.g., ISOLET, devote the majority of the energy/execution time to perform the associative search. Our evaluation shows that F5-HD achieves 11.9× and 1.7× (616.8× and 259.9×) higher energy efficiency and faster inference as compared to GPU (CPU) platform respectively.

F5-HD can have different design choices for inference. Using fixed-point module F5-HD provides the maximum classification accuracy but relatively slower computation. Using binary and power-of-two model, the encoding dominates F5-HD energy/execution time, while for the fixed-point model the majority of resources are devoted to the associative search. F5-HD removes the multiplications involved in cosine similarity using power-of-two model, resulting in higher computation efficiency. Finally, the binary model is the most efficient F5-HD model, where the similarity check can be performed by using Hamming distance. Figure 7 shows the F5-HD inference efficiency using power-of-two and binary models. All results are normalized to the throughput and throughput/Watt of F5-HD with fixed-point model. For applications with low feature size, e.g., PAMAP, the encoding module maps a large number of data points into high-dimensional space. This makes the associative search a dominant part of inference computation when using fixed-point model. On the other hand, in face detection with a low number of classes and high feature size, the encoding dominates the F5-HD resource and efficiency. Our evaluation shows that F5-HD using binary and power-of-two models can achieve on average 4.3× and 3.1× higher throughput than F5-HD using fixed-point model. In addition, the binary and power-of-two models provide

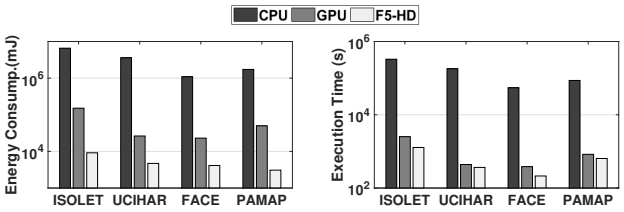


Figure 6: Energy consumption and execution time of HD during inference running on different platforms.

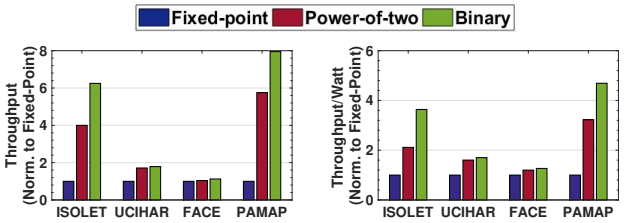


Figure 7: Throughput and throughput/Watt in F5-HD using fixed-point, power-of-two, and binary models.

Table 3: Average resource utilization and power consumption of F5-HD implemented on Kintex

		Fixed-point	Power-of-two	Binary
Resource utilization	LUT	46%	95%	82%
	BRAM	47%	46%	5%
	DSP	89%	26%	29%
Power(W)		9.8	14.8	13.9

2.1× and 1.5× higher throughput/Watt as compared to F5-HD using fixed-point model.

5.4 Resource/Power Utilization

Table 3 lists the average Kintex FPGA resource utilization implementing F5-HD using fixed-point, power-of-two, and binary models. The results are reported for F5-HD supporting both training and inference. Our evaluation shows that the fixed-point model utilizes the majority of the FPGA DSPs in order to perform the similarity check of the inference/retraining. In contrast, with binary and power-of-two models have much lower DSP utilization, as the majority of their inference computation includes bitwise operations that can be efficiently performed using LUTs and the PopCounter. In addition, F5-HD with the binary model has the lowest BRAM utilization as it can store the trained HD model using significantly lower memory size. Table 3 also provides the average power dissipation of the Kintex FPGA. The results indicate that in the fixed-point model, the number of DSPs limits the FPGA throughput, thus F5-HD consumes lower power consumption due to its overall low LUT utilization. In contrast, F5-HD using binary model highly utilizes the available LUTs on the FPGA resulting in high throughput and higher power consumption.

5.5 F5-HD on Different FPGA Platforms

To demonstrate the generality of F5-HD and further investigate its efficiency, we implement it on three different FPGA platforms, mentioned earlier in this section. Figure 8(a) compares the average throughout of F5-HD running different HD applications on these three platforms. Our evaluation shows that Virtex implementing fixed-point model provides 12.0× and 2.5× higher throughput as compared to Spartan and Kintex platforms. The efficiency of Virtex comes from its large amount of available DSPs (2,800 DSPs with 485K LUTs), which can be used to accelerate associative search. However, F5-HD using power-of-two and binary models mostly exploit LUTs for FPGA implementation, resulting in higher throughput especially on Spartan with few numbers of DSPs. For example, Spartan using binary model can achieve on average 5.2× higher throughput than F5-HD using fixed-point model. It should be noted that in all FPGA platforms the throughput of the binary model is proportional to the number of available LUTs in FPGAs.

To compare the computation efficiency of different FPGAs, we eliminate the impact of available resources by using the throughput/Watt as the comparison metric. Figure 8(b) shows the throughput/Watt of F5-HD implemented in different platforms. As the results show, Virtex with large number of DSPs provides the maximum throughput/Watt when implementing F5-HD using fixed-point model. However, using power-of-two and binary models, Spartan provides the higher computation efficiency since most of F5-HD computation can be processed by LUTs. For example, using the fixed-point model, Virtex can provide 2.0× and 1.5× higher throughput/Watt as compared to Spartan and Kintex, respectively. However, using the binary model, Spartan provides 1.2× and 1.5× higher throughput/Watt than Virtex and Kintex respectively.

The efficiency of different FPGAs also depends on the application, i.e., number of features and classes. For applications with small feature size (e.g., PAMAP), F5-HD can encode a larger amount of data at a time, thus the associative search in inference requires higher number of DSPs and BRAM accesses to parallelize the similarity check. This makes the number of DSPs the bottleneck of computation when using a fixed-point model for PAMAP application. PAMAP using power-of-two model eliminates the majority of DSP utilization required to multiply a query and class hypervector, thus the number of BRAMs becomes the computation bottleneck. These results are more obvious on the Spartan FPGA with limited BRAM blocks.

5.6 Power Budget

As we explained in Section 3, the desired power budget is an input to F5-HD framework that can be dictated by the users before implementation of each application, which impacts the level of parallelism. When the user defines a desired power budget (P_{target}), F5-HD tries to determine the number of PEs per PU such that the implementation satisfies the power constraint. In practice, F5-HD may not precisely guarantee the desired power due to the fact that the number of PEs per PU has discrete values and the size of the application and its power consumption depend on this discrete parameter. Additionally, our initial estimation of the power consumption is according to the logical connectivity of the building blocks and may not accurately estimate the impact of signals power, which is routing-dependent[2]. Therefore, the measured power after implementation (P_{meas}) might have fluctuations around the target power level. Here we define the power fluctuation as $\Delta P = {|P_{meas} - P_{target}|}/{P_{target}}$.

Table 4 lists the average throughput (TP) and ΔP after imposing the power budget. The table also shows the normalized throughput under power constraints to the nominal throughput when no power budget is employed. The results are reported for the cases that the power budget is defined as 25% and 50% of maximum power (power of F5-HD running on the same device without power restriction) as the desired power level. Our evaluations show that our framework

[2]In practice, we scale the power of the signal based on the measured signal power of a base implementation

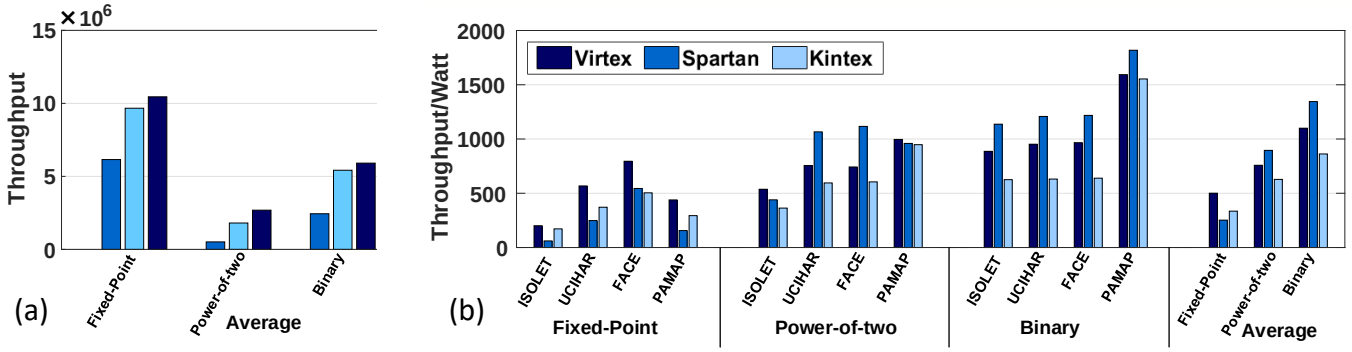


Figure 8: (a) Average throughput of different FPGAs implementing F5-HD with fixed-point, power-of-two, and binary models. (b) Throughput/Watt of F5-HD implementing different applications on FPGA platforms.

Table 4: F5-HD implementation under power constraints.

FPGA model	Power budget	Fixed-point TP	Fixed-point ΔP	Power-of-two TP	Power-of-two ΔP	Binary TP	Binary ΔP
Virtex	50%	2.7 (0.44×)	5.2%	5.1 (0.53×)	5.2%	6.3 (0.60×)	2.4%
	25%	1.2 (0.19×)	10.8%	1.6 (0.16×)	7.0%	2.6 (0.25×)	8.0%
Spartan	50%	0.2 (0.38×)	12.8%	0.7 (0.37×)	5.1%	1.0 (0.40×)	5.1%
	25%	0.1 (0.21×)	9.1%	0.3 (0.17×)	16%	0.6 (0.21×)	17%
Kintex	50%	1.0 (0.41×)	5.2%	2.5 (0.45×)	6.8%	4.8 (0.65×)	4.0%
	25%	0.4 (0.18×)	12.1%	1.3 (0.24×)	18%	1.7 (0.25×)	12%

can generate HD accelerator that lays within ΔP = 18% of the target power. The power fluctuation becomes large when the targeted power is low as the magnitude of misprediction ($|P_{meas} - P_{target}|$) almost remains the same while the base power P_{target} reduces.

6 CONCLUSION

In this paper, we proposed F5-HD, an automated framework for FPGA-based acceleration of HD computing. F5-HD abstracts away the complexities behind designing hardware accelerators from the user. The proposed framework enables the user to specify the HD application specifications (e.g., the number of input features, classes and training data) as well as the desired classification quality (i.e., accuracy versus performance) and accordingly generates customized FPGA-friendly Verilog implementation. In addition to training and inference, F5-HD supports simultaneous training and inference, hence the accuracy of the HD platform can be enhanced in the field without, without interrupting its operation. We evaluated the efficiency of F5-HD extensively, whereby it showed 86.9× and 7.8× (11.9× and 1.7×) higher energy efficiency improvement and faster training (inference) as compared to an optimized implementation of HD on AMD R9 390 GPU, respectively.

ACKNOWLEDGEMENTS

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

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] P. Kanerva, "Computing with 10,000-bit words," in *Communication, Control, and Computing (Allerton), 2014 52nd Annual Allerton Conference on*, pp. 304–310, IEEE, 2014.
[3] A. Rahimi, P. Kanerva, and J. M. Rabaey, "A robust and energy-efficient classifier using brain-inspired hyperdimensional computing," in *Proceedings of the 2016 International Symposium on Low Power Electronics and Design*, pp. 64–69, ACM, 2016.
[4] F. R. Najafabadi, A. Rahimi, P. Kanerva, and J. M. Rabaey, "Hyperdimensional computing for text classification," in *Design, Automation Test in Europe Conference Exhibition (DATE), University Booth*, pp. 1–1, 2016.
[5] O. J. Räsänen and J. P. Saarinen, "Sequence prediction with sparse distributed hyperdimensional coding applied to the analysis of mobile phone use patterns," *IEEE transactions on neural networks and learning systems*, vol. 27, no. 9, pp. 1878–1889, 2016.
[6] M. Imani, D. Kong, A. Rahimi, and T. Rosing, "Voicehd: Hyperdimensional computing for efficient speech recognition," in *Rebooting Computing (ICRC), 2017 IEEE International Conference on*, pp. 1–8, IEEE, 2017.
[7] F. Montagna, A. Rahimi, S. Benatti, D. Rossi, and L. Benini, "Pulp-hd: accelerating brain-inspired high-dimensional computing on a parallel ultra-low power platform," in *Proceedings of the 55th Annual Design Automation Conference*, p. 111, ACM, 2018.
[8] 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.
[9] A. Joshi, J. Halseth, and P. Kanerva, "Language geometry using random indexing," *Quantum Interaction 2016 Conference Proceedings*, In press.
[10] 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.
[11] S. Salamat, M. Imani, S. Gupta, and T. Rosing, "Rnsnet: In-memory neural network acceleration using residue number system," in *Rebooting Computing (ICRC), 2018 IEEE International Conference on*, pp. 1–10, IEEE, 2018.
[12] 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.
[13] 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.
[14] M. Imani, A. Rahimi, D. Kong, T. Rosing, and J. M. Rabaey, "Exploring hyperdimensional associative memory," in *2017 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pp. 445–456, IEEE, 2017.
[15] A. DeHon, "The density advantage of configurable computing," *Computer*, vol. 33, no. 4, pp. 41–49, 2000.
[16] S. Salamat, M. R. Azarbad, and B. Alizadeh, "Improve high level synthesis for multi-dimensional nested loops using reshaping and vectorization methods for multi-level non-rectangular nested loop," in *Rebooting Computing (ICRC), 2018 IEEE International Conference on*, pp. 1–10, IEEE, 2018.
[17] 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.
[18] M. Imani *et al.*, "Low-power sparse hyperdimensional encoder for language recognition," *IEEE Design & Test*, vol. 34, no. 6, pp. 94–101, 2017.
[19] M. Imani *et al.*, "Hdna: Energy-efficient dna sequencing using hyperdimensional computing," in *BHI*, pp. 271–274, IEEE, 2018.
[20] Y. Kim *et al.*, "Efficient human activity recognition using hyperdimensional computing," in *IoT*, p. 38, ACM, 2018.
[21] M. Imani *et al.*, "Fach: Fpga-based acceleration of hyperdimensional computing by reducing computational complexity," in *ASP-DAC*, IEEE, 2019.
[22] M. Imani *et al.*, "A binary learning framework for hyperdimensional computing," in *DATE*, IEEE/ACM, 2019.
[23] M. Imani *et al.*, "Hdcluster: An accurate clustering using brain-inspired high-dimensional computing," in *DATE*, IEEE/ACM, 2019.
[24] T. F. Wu, H. Li, P.-C. Huang, A. Rahimi, J. M. Rabaey, H.-S. P. Wong, M. M. Shulaker, and S. Mitra, "Brain-inspired computing exploiting carbon nanotube fets and resistive ram: Hyperdimensional computing case study," in *Solid-State Circuits Conference-(ISSCC), 2018 IEEE International*, pp. 492–494, IEEE, 2018.
[25] H. Li *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.
[26] S. Gupta *et al.*, "Felix: fast and energy-efficient logic in memory," in *ICCAD*, p. 55, ACM, 2018.
[27] B. Falsafi, B. Dally, D. Singh, D. Chiou, J. Y. Joshua, and R. Sendag, "Fpgas versus gpus in data centers," *IEEE Micro*, vol. 37, no. 1, pp. 60–72, 2017.
[28] "Xilinx power estimator user guide." User Guide, June 2017.
[29] T. Feist, "Vivado design suite," *White Paper*, vol. 5, 2012.
[30] "Amd." http://developer.amd.com/tools-and-sdks/opencl-zone/codexl/.
[31] "Uci machine learning repository." http://archive.ics.uci.edu/ml/datasets/ISOLET.
[32] "Uci machine learning repository." https://archive.ics.uci.edu/ml/datasets/Daily+and+Sports+Activities.
[33] A. Reiss and D. Stricker, "Creating and benchmarking a new dataset for physical activity monitoring," in *Proceedings of the 5th International Conference on PErvasive Technologies Related to Assistive Environments*, p. 40, ACM, 2012.
[34] G. Griffin, A. Holub, and P. Perona, "Caltech-256 object category dataset," 2007.
[35] M. Everingham, S. A. Eslami, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes challenge: A retrospective," *International journal of computer vision*, vol. 111, no. 1, pp. 98–136, 2015.