2021 IEEE 39th International Conference on Computer Design (ICCD) | 978-1-6654-3219-1/21/$31.00 ©2021 IEEE | DOI: 10.1109/ICCD53106.2021.00026

AdaptBit-HD: Adaptive Model Bitwidth for Hyperdimensional Computing

Justin Morris*†, Si Thu Kaung Set*, Gadi Rosen*, Mohsen Imani‡, Baris Aksanli†, and Tajana Rosing*
*University of California San Diego, La Jolla, CA 92093, USA
†San Diego State University, San Diego, CA 92182, USA
‡University of California Irvine, Irvine, CA 92697, USA
{justinmorris,skaungse,grosen}@ucsd.edu, m.imani@uci.edu, baksanli@sdsu.edu, tajana@ucsd.edu

***Abstract*—Brain-inspired Hyperdimensional (HD) computing is a novel computing paradigm emulating the neuron's activity in high-dimensional space. The first step in HD computing is to map each data point into high-dimensional space (e.g., 10,000). This poses several problems. For instance, the size of the data can explode and all subsequent operations need to be performed in parallel in D = 10,000 dimensions. Prior work alleviated this issue with model quantization. The HVs could then be stored in less space than the original data and lower bitwidth operations can be used to save energy. However, prior work quantized all samples to the same bitwidth. We propose, AdaptBit-HD, an Adaptive Model Bitwidth Architecture for accelerating HD Computing. AdaptBit-HD operates on the bits of the quantized model one bit at a time to save energy when fewer bits can be used to find the correct class. With AdaptBit-HD, we can achieve both high accuracy by utilizing all the bits when necessary and high energy efficiency by terminating execution at lower bits when our design is confident in the output. We additionally design an end-to-end FPGA accelerator for AdaptBit-HD. Compared to 16-bit models, AdaptBit-HD is $14\times$ more energy efficient and compared to binary models, AdaptBit-HD is 1.1% more accurate, which is comparable in accuracy to 16-bit models. This demonstrates that AdaptBit-HD is able to achieve the accuracy of full precision models, with the energy efficiency of binary models.**

I. Introduction

Brain-inspired Hyperdimensional computing has been proposed as a light-weight algorithm to perform cognitive tasks at significantly less computational complexity than Neural Networks (NN). Inspired by research from neuroscience, HD computing represents data as points in a high dimensional space. Past research utilized high dimension vectors (D (Dimensionality) $\geq$ 10,000), called hypervectors (HV), to represent neural activity in the brain [1]. There are several nearly orthogonal HVs in a high-dimensional space [2]. This enables HD to combine HVs with well-defined vector operations, while preserving most of the information from each individual HV to create a model for inference. The number of operations during inference in HD is in the order of 10^5 [3]. This is significantly less compared to DNNs, with the amount of operations in inference reaching over 10^9 [4]. HD Computing also learns fast, only needing a few passes over the training data, compared to DNNs, which need hundreds or thousands of retraining iterations [5]. These properties of HD make it suitable for small, resource limited devices.

In HD computing, training data points are combined into a set of hypervectors, called an *HD model*, through light-weight computation steps. Each hypervector in the model represents a class of the target classification problem. Most of the proposed HD computing work exploits binarized hypervectors to reduce the computational/memory intensity in HD computing [6], [7]. Although binary hypervectors lead to significantly more energy efficient models, there is an accuracy cost for moving to the binary domain. To alleviate this, there has also been work on quantizing HD models to different bitwidths [8]. However, the existing HD computing quantization methods have two main challenges: (i) the trade-off between accuracy and energy efficiency has to be decided before training the model, and the model would have to be retrained from scratch to change bitwidths if the accuracy and energy efficiency trade-off requirements change. (ii) Existing model quantization techniques ignore that not all samples need to be quantized with the same value. Some samples can be classified with simple binary representations, while others require higher bitwidths for accurate classification. In other words, there exists no adaptive bitwidth quantization for HD Computing. Adaptive bitwidth quantization adds another level of tuning for systems balancing the accuracy and energy efficiency trade-off.

In this paper, we propose AdaptBit-HD, which, to the best of our knowledge, is the the first Adaptive Model Bitwidth Architecture for accelerating HD computing. AdaptBit-HD does not change the bitwidth of the representation of the data, but operates on the bits of the quantized model one bit at a time to save energy when fewer bits can be used to find the correct class. AdaptBit-HD can achieve both high accuracy by utilizing all bits when necessary and high energy efficiency and faster execution time by terminating execution at lower bits when our design is confident in the output. AdaptBit-HD achieves this by performing a hamming distance operation on the query HV and class HVs one bit at a time. We check after each bit if we are confident enough in our current answer to terminate execution early. To achieve this, we completely redesign the HD computing algorithm including training, retraining, and inference. We accordingly design an end-to-end HD FPGA accelerator for AdaptBit-HD and compare with a state-of-the-art binary quantization FPGA accelerator for HD [9] as well as a 16-bit static quantization method. Compared to binary quantization AdaptBit-HD is 1.1% more accurate at the cost of just 10% more energy consumption and 7% more execution time. Compared to 16-

978-1-6654-3219-1/21/$31.00 ©2021 IEEE
DOI 10.1109/ICCD53106.2021.00026

bit models, AdaptBit-HD is $14\times$ more energy efficient at the cost of 0.1% accuracy.

II. Related Work

Model quantization is a widely used technique in machine learning to improve energy efficiency. For instance, Google's TPU for performing inference on DNNs utilizes reduced bit representations [10]. Model quantization has also been used to reduce the memory requirement for a more efficient hardware design [11]. Other methods such as model compression have also been used to improve the energy efficiency of neural networks [12]. Model quantization has also been widely used to accelerate and improve the energy consumption of HD computing [8], [13]. Although model quantization has improved other machine learning methods such as DNNs, light-weight models such as HD Computing continue to be more energy efficient and for applications where energy efficiency is paramount, light-weight models should continue to be utilized. Work has also been done to adaptively change the precision of the model to reduce the accuracy loss online [14]. [15] proposes a method to use multiple precision levels during inference to achieve a balance between efficiency and accuracy loss. [16] tries to alleviate accuracy loss from quantization by compensating for computational errors.

Prior work on HD Computing has also shown that quantizing the class hypervectors can provide significant energy and speedup improvements at the cost of accuracy [13], [17]. Work in [8] extended the idea of binarizing the class HV weights to using a ternary model to achieve higher accuracies. However, all of the existing work on HD Computing for model quantization is static. This poses a few problems. For instance, if accuracy and energy efficiency needs change, the model needs to be completely retrained to change bitwidths. Additionally, by being static, one has to choose where they land on the accuracy and energy trade-off curve at a macro level. This often leads to leaning towards one end of the spectrum, either highly energy efficient with accuracy loss, or highly accurate with high energy consumption. This problem is exaggerated for applications with varying precision needs based on the incoming data. However, in this paper, we propose an adaptive bitwidth quantization method that chooses the best bitwidth per sample to achieve a confident classification with minimal energy usage. This leads to an overall design that achieves both high accuracy and high energy efficiency.

III. AdaptBit-HD

We propose AdaptBit-HD, the first Adaptive Model Bitwidth Architecture for accelerating HD computing. HD computing consists of three main modules shown in Figure 1: encoding, training, and inference. The encoding module maps each data point to high-dimensional space. The HD model accumulates every encoded training hypervector (HV) to create an integer model. This integer model is then used to create a quantized model. During inference, HD computing then chooses the most similar class to the query HV as the output class. AdaptBit-HD fundamentally changes the

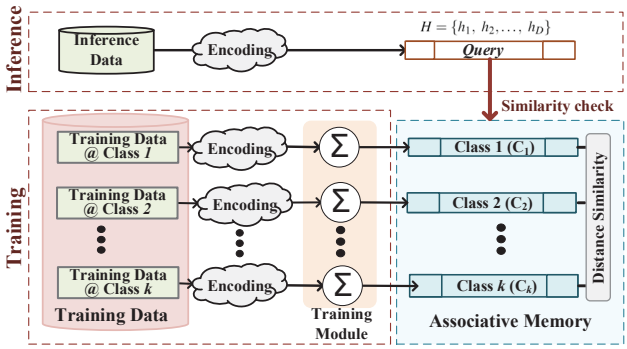

Fig. 1. Overview of creating an HD model and performing inference.

inference phase by operating on the bits of the quantized model one bit at a time to save energy when fewer bits can be used to find the correct class. We check after each bit if we are confident enough in our current answer to terminate execution early based on a threshold of similarity. By operating with this new inference technique, AdaptBit-HD is able to achieve the energy efficiency of binary models, while maintaining the accuracy of full precision models. To further support our proposed inference design, AdaptBit-HD accordingly proposes a training approach that trains the model to create quantized HVs and tunes the model to improve the confidence of the threshold we utilize to determine if we can terminate execution early. In the following section, we explain the details of the baseline HD functionality and AdaptBit-HD.

A. Baseline Encoding

The encoding first replicates the feature vector, $\mathbf{F}$, extending it to D dimensions, the same as our desired high-dimensional vector. For example, to encode a feature vector with $n = 500$ features to $D = 4,000$ dimensions, we need to concatenate 8 copies of a feature vector together. Then, it generates a random D dimensional projection vector, P. To compute the dimensions of the high-dimensional vector, we take the dot product of the extended feature vector with each projection vector in an N-gram window. The first N-gram is the dot product of the first N features and N projection vector elements:

$$h_1 = sign(f_1 * p_1 + f_2 * p_2 + ... + f_N * p_N)$$

Similarly, the N-gram window shifts by a single position to generate the next feature values. So, we can compute the i^{th} dimension of an encoded hypervector using:

$$h_i = sign(f_i * p_i + f_{i+1} * p_{i+1} + ... + f_{i+N} * p_{i+N})$$

B. Baseline Training and Retraining

HD computing supports efficient one-pass training. To build a one-pass model, the encoder maps all training data to training HVs ($\mathcal{H}$). For all training HVs within a class ($\{\mathcal{H}_i^1, \mathcal{H}_i^2, \ldots, \mathcal{H}_i^j\}$), HD computing adds them together to create a single class HV ($\mathcal{C}_i$).

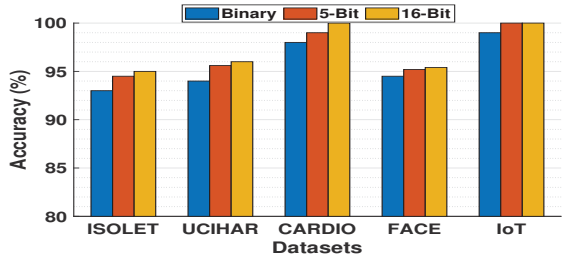

Fig. 2. Difference in Accuracy with Various HD Bitwidth Representations for HD Computing

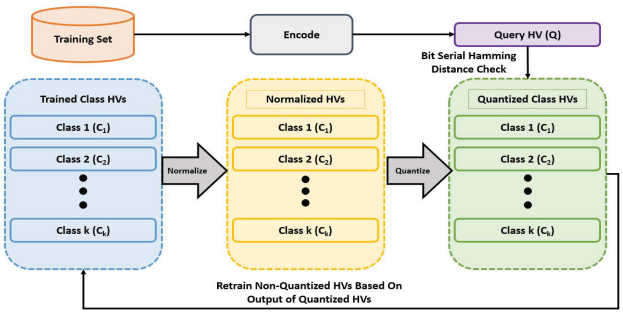

Fig. 3. Overview of Creating a AdaptBit-HD Model During Retraining

$$\mathcal{C}_i = \mathcal{H}_i^1 + \mathcal{H}_i^2 + \ldots + \mathcal{H}_i^j$$

Once this is done for every class, we have an HD model that can be used for inference. However, we can significantly improve the accuracy of our HD model with retraining [18]. We retrain the HD model by inputting each training data point through the HD model as a query hypervector (Q); (i) if the query is correctly classified by the current model, our design does not change the model. (ii) If it is incorrectly matched with the i^th class hypervector (C_i), when it actually belongs to j^th class (C_j), the retraining procedure subtracts the query hypervector from the i^th class and adds it to j^th class hypervector: $C_i = C_i - Q$ and $C_j = C_j + Q$.

C. Training with AdaptBit-HD

This section explains how we achieve this adaptive model quantization with HD computing. Figure 2 demonstrates the idea that not all samples need the same bitwidth to be accurately labeled. For instance, in the figure, we can see that HD computing is able to achieve an average of 95.7% accuracy with binary values. Additionally, by moving to a $5-bit$ representation, HD is able to improve in accuracy by 1.16% on average. Comparing the $5-bit$ representations with full precision, we can see that the 16-bit precision model is only able to achieve 0.42% more accuracy than the $5-bit$ models. This demonstrates that for most samples, we can get away with aggressive model quantization. However, there are some samples that require more bits to separate the data properly to maintain high accuracy. Rather then using high precision for all of the samples to achieve high accuracy, we can adaptively select the bitwidth we need for the sample during inference to balance both accuracy and energy efficiency.

Initial Training: The initial training for AdaptBit-HD first builds the full bitwidth model by combining all samples as described in III-C. The training process for model quantization diverges from that of past work after the initial training. As Figure 3 shows, we first normalize all class HVs such that all of the dimensions are in the range [-1,1], but we still keep the non-normalized vectors. We then quantize the normalized vectors to the nearest power of 2 in a list of quantized values.

The list of powers of two is defined by two parameters: n, the number of bits, and o, the offset of the powers of two. The offset is to control where in the range of values $(-1, 1)$ we want to have a higher resolution of representation. Higher offsets lead to better quantization near 0. We first set aside one bit for representing 0. Then the rest of the $2^{(n-1)}$ representations are as follows: $2^{(r-o)}$ where r = -1, -2, ..., $-(2^{(n-2)})$ and o is the offset. We iterate r $2^{(n-2)}$ times because we additionally represent the same powers of two on the negative side. Each power of two representation is then assigned a unique $n-bit$ binary string. For example, if $n = 3$ and $o = 0$, we would be able to represent the following powers of 2 in our model quantization: (-2^{-1}, -2^{-2}, 0, 2^{-1}, 2^{-2}), where each one of these values is assigned a unique $3-bit$ sequence of $0s$ and $1s$. Once we set each dimension to the closest power of two we can represent, we then have an HD model where each dimension is an $n-bit$ value.

To encode our weights to unique $n-bit$ values, we assign the first bit to indicate if the value is negative or positive. This ensures that the first bit hamming distance is equivalent to how binary models are created for HD. We reserve the second bit to indicate if the value is zero or not. Thus, the calculation of the hamming distance for the second bit is equivalent to counting the number of matching zeros. For the rest of the bits, because we use hamming distance as our similarity metric (which is explained in Section III-E) on the binary representation of our values instead of the cosine similarity of the values themselves, it is important that values near each other have a small hamming distance score between each other. To achieve this, we use a grey code encoding to assign the last bits of the binary strings to each power of 2.

Retraining: To retrain AdaptBit-HD, we store both a full precision model and an $n-bit$ representation model of the class HVs. We retrain the quantized model by iterating through the training set. In a single iteration of model adjustment, AdaptBit-HD checks the similarity of all training data points, say $\mathbf{H}$, with the class HVs in the quantized model. If the datapoint is correctly classified, normally, no model update is needed. However, in Section III-E, we modify this to support adaptively stopping the similarity check in a bit-serial manner. If a data point is incorrectly classified by the model, HD updates the model by (i) adding the incorrectly classified HV to the class the input data point belongs to ($\widetilde{\mathbf{C}}^{correct} = \mathbf{C}^{correct} + \mathbf{H}$), and (ii) subtracting it from the class to which it is wrongly matched ($\widetilde{\mathbf{C}}^{wrong} = \mathbf{C}^{wrong} - \mathbf{H}$). These changes are made to the full precision model saved from

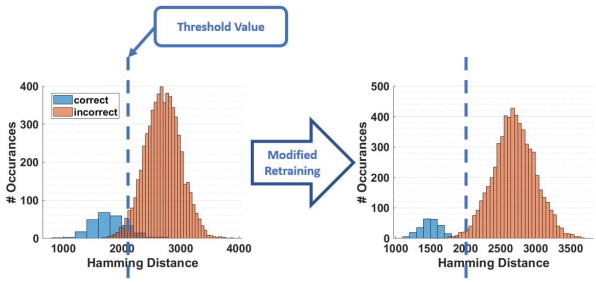

Fig. 4. Distributions of Hamming Distance Calculations Before (left) and After (right) Retraining

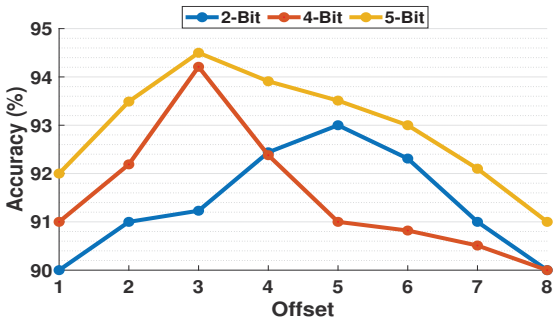

Fig. 5. Effect of AdaptBit-HD Parameters on Accuracy

training because adding to and subtracting from the quantized model would drastically change the model. To update the quantized model, the updated class HV from the integer model are quantized via the same process described in Section III-C.

D. Baseline Inference

Upon inference, the encoder first maps the input data into a query HV ($\mathcal{Q}$), using the same encoding to train the HD model. A similarity metric is used to determine the strength of a match between the query HV and each class HV. The most common metric used in HD computing is cosine similarity, but other metrics (e.g. Hamming distance) could be appropriate depending on the problem [3]. After the similarity is computed between the query HV and each class HV in the classifier, the class with the highest similarity is chosen as the output class.

E. Inference with AdaptBit-HD

To support our new hamming distance check, the query is quantized the same way as the class HVs. Then, AdaptBit-HD calculates the hamming distance between the first bit of the class HVs and the query HV across all the dimensions. We then check to see if the class with the highest similarity passes a threshold value. If the similarity threshold is passed, then execution can stop prematurely and output the current highest similarity class. However, if the threshold is not passed, then computation continues to the next bit and the hamming distances are accumulated. We then check the if the similarity threshold is met again and if it is not, we continue the process. If the similarity threshold is not met and we are on the last bit, the most similar class is the output.

Thresholding: In order to support the one bit at a time hamming distance, we need to create a threshold for the termination condition. To do this, after the initial training, for the first iteration of retraining, we collect the hamming distance for all samples and the class HVs. As hamming distance calculates the number of mismatches, the incorrect samples should be clustered with higher hamming distance values and the correct samples should be clustered in a lower distribution. We then get the mean and standard deviation of all samples where our model was correct as well as the mean and standard deviation of all samples where our model was incorrect. We next set the threshold to be the average between the $mean$ - $standard\ deviation$ of the incorrect distribution and the $mean$ + $standard\ deviation$ of the correct distribution. This heuristic makes sense to use because averaging in this way gives us a good initial threshold that separates the two distributions. We do this process per class as the distribution of similarity values differs on a per class basis, thus, we need a different threshold for each class.

Figure 4 shows the distribution of hamming distances for all samples for a single class. The graph on the left shows our initial threshold, where the initial distributions are not completely separated by the threshold. To fix this, we modified the retraining algorithm to actively create a greater separation in order to minimize outputting the incorrect class when the threshold is met. To do this we made the following change: if the datapoint is correctly classified, rather than doing nothing, because we are correct, we additionally check if the similarity threshold was met. If the threshold is met with an additional 10% guard-band, we do nothing. However, if this is not met, we add the query hypervector to the class the input data point belongs to ($\widetilde{\mathbf{C}}^{correct} = \mathbf{C}^{correct} + \mathbf{H}$). As Figure 4 shows on the graph to the right, after retraining the distribution of incorrectly classified and correctly classified samples are further separated leading to more accurate classification when we terminate the hamming distance operation early.

The 10% guard-band is to help ensure we push the distribution of correct samples past the threshold value. This leads to a more accurate model when terminating early based on the threshold. As Figure 4 shows, when the hamming distance passes the threshold, we can be confident that it is the correct class.

Figure 5 shows the impact of using different offsets and bitwidths for AdaptBit-HD on the ISOLET dataset, however, the results are similar for all datasets tested. As the figure shows, there is a balance between having too high and too low of an offset. This is because with too low of an offset, we have less quantization resolution near 0. However, with too high of an offset, we again will not have enough resolution at the ends of our distribution ($[-1, 1]$). The figure shows that generally, an offset of 3 gives the best balance in quantizing the range of values. The exception is when using 2 bits as with so few bits, we can only choose to have good resolution at either the ends of the distribution or near 0. It turns out

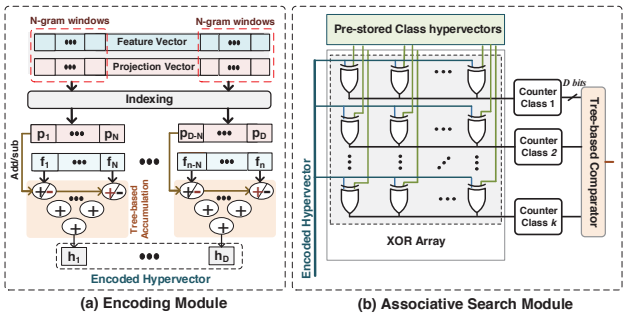

Fig. 6. FPGA implementation of the encoding and associative search block.

that having higher resolution near 0 is more important leading to higher accuracies with higher offsets when only using 2 bits. Additionally, we saw that accuracy becomes saturated at 5 bits, and is comparable to a model with no quantization in Figure 2. This stayed consistent across all datasets tested.

F. FPGA Acceleration

AdaptBit-HD can be accelerated on different platforms such as CPU, GPU, FPGA, or ASIC. FPGA is one of the best options as AdaptBit-HD computation involves bitwise operations among long vector sizes, e.g., encoding and associative search. Additionally, unlike ASICs or PIM implementations, FPGAs offer reprogramability and faster design times. General strategies of optimizing the performance of AdaptBit-HD are (i) using a pipeline and partial unrolling on a low level (dimension level) to speed up each task and (ii) using dataflow design on a high level (task level) to build a stream processing architecture that lets different tasks run concurrently. In the following, we explain the functionality of AdaptBit-HD in encoding, training, retraining, and inference phases.

Encoding Implementation: As in Section III-A, we used the locality-based random projection encoding to implement the encoding module. Due to the sequential and predictable memory access patterns as well as the abundance of binary operations, this encoding approach can be implemented efficiently on an FPGA. In the hardware implementation, we represent all $\{-1, +1\}$ values with $\{0, 1\}$ respectively. This enables us to represent each element of the projection vector using a single bit. Figure 6a shows the hardware implementation of the AdaptBit-HD encoding module. Calculating the inner product of a feature vector and a projection vector, $P \in \{1, -1\}^D$, can be implemented with no multiplications. Each element of the projection vector decides the sign of each dimension of the feature vector in the accumulation of the dot product. Thus, the dot product can be simplified to the addition/subtraction of the feature vector elements. We use Look Up Tables (LUTs) and Flip Flops (FFs) resources of the FPGA to implement the encoding module, rather than DSPs for this addition, which leads to better energy efficiency.

Initial Training: Like previously, initial training for AdaptBit-HD with model quantization is a single-pass process through the training dataset. The training module accesses the encoded hypervectors and accumulates them in order to create a hypervector representing each class. We exploit data-flow design implementing the encoding and initial training modules in a pipeline structure. When the training module accumulates the encoded hypervector to one of the class hypervectors, the encoding module maps the next training data into high-dimensional space, improving data throughput by increasing resource utilization. This improves FPGA throughput by maximizing resource utilization as well as hides the latency of the encoding. After going through all of the training data, our implementation creates an n-bit quantized representation of the model. This process is described in s described in Section III-C. The quantized n-bit model is stored in the BRAM blocks to be used for inference or retraining.

We first normalize all of the class HVs such that all of the dimensions are in the range [-1,1], but we still keep the non-normalized class HVs for use during retraining. We then quantize the normalized vectors to the nearest power of 2 in a list of quantized values as described in Section III-C. Finally, the quantized n-bit model is stored in the BRAM blocks to be used for inference or retraining. Since the generating and writing are done only once during the entire training process, they do not impact the performance of the training phase. Thus, these two parts are not fully optimized, allowing our design to saves some resources for the retraining phase, which is more critical to the overall performance.

Retraining: Retraining is implemented separately from training, since the final result of initial training, the n-bit model, will be used in retraining, so they are performed sequentially. The retraining phase first sequentially reads already encoded training hypervectors from the off-chip memory in batches to help hide the latency of reading from the off-chip memory. This is necessary as each read has a latency of about $15ns$, which would slow down the retraining process. Next, we check the hamming distance similarity of each data point with all trained class hypervectors. As mentioned in Section III-E, this is performed in a bit serial fashion. To support this in hardware we split the hamming distance calculation into its own pipeline stage. If the threshold is met, then the pipeline continues. However, if the threshold is not met and we need to go to the next bit, the pipeline is stalled and the next bit is calculated. This process continues until the threshold is met or we are out of bits to process. This may look like a large performance impact by stalling the pipeline, however, as the experimental results section demonstrates, over 90% of all samples terminate at just one bit, so we do not need to stall the pipeline often. At the end, each data point gets a tag of the class in which it has the highest Hamming distance.

The Hamming distance similarity check is implemented using an `XOR` array which compares the bit similarity between two hypervectors. Counter blocks, shown in Figure 6b, calculate the number of mismatches of each class hypervector with the query data point. Finally, a tree-based comparator block finds the class with the lowest counter value. In the case of misclassification, AdaptBit-HD needs to update the model by adding and subtracting a data hypervector with two class hypervectors as defined before. Like encoding, all retraining

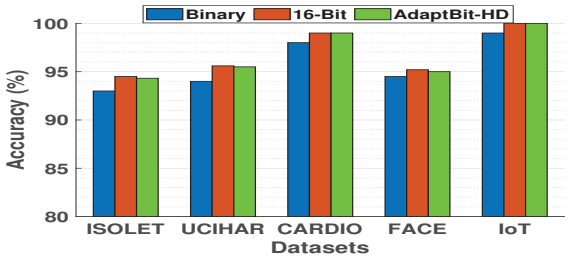

Fig. 7. Comparison of the Accuracy of AdaptBit-HD to Static Model Quantization Methods

processes can be implemented using LUTs and FFs blocks.

The uniqueness of the retraining implementation in our design is that we have two different models in similarity check and updating. This avoids the situation of similarity check and updating access to the same model simultaneously, which makes the dataflow design in a single iteration possible. After certain batch size iterations, the process stream is stopped for a while and the model in similarity check will be refreshed by the model in updating. This design will bring $2\times$ better speed and little influence on the result of retraining.

Inference Implementation: After the retraining, the quantized AdaptBit-HD model has a stable model that can be used in the inference phase. The encoding module is integrated with the similarity check module used during retraining as the entire inference part. Each test data point is first encoded to high-dimensional space using the same encoding block explained in Section III-F. Next, the quantized AdaptBit-HD model checks the Hamming distance similarity of the data point with all pre-stored class HVs, in a bit serial manner, in order to find a class with the highest similarity.

IV. Evaluation

A. Experimental Setup

We implemented AdaptBit-HD training, retraining, and inference in both software and hardware. In software, we implemented AdaptBit-HD with Python. In hardware, we fully implemented AdaptBit-HD using Verilog. We verify the timing and the functionality of the models by synthesizing them using Xilinx Vivado Design Suite [19]. The synthesis code has been implemented on the Kintex-7 FPGA KC705 Evaluation Kit and we used the Vivado XPower tool to estimate the device power [20]. We compare AdaptBit-HD with baseline HD, an FPGA implementation of [9] using a binary model. We additionally compare to a static 5-bit HD computing design. For all of our experiments, we use $D = 4,000$, a bitwidth of 5, and offset of 3 as these parameters experimentally showed the best performance across all datasets.

We evaluated the efficiency of AdaptBit-HD on four practical classification problems listed below: Speech Recognition (ISOLET) [21], Activity Recognition (UCIHAR) [22], Face Detection (FACE) [23], Cardiotocography (CARDIO) [24], and Attack Detection in IoT systems (IoT) [25].

TABLE I
Comparison of the Average Accuracy of Different Bit Representations vs AdaptBit-HD

Dimensionality	4,000	2,000	1,000	800	500
Binary	95.70%	95.64%	94.60%	93.14%	89.13%
5-bit Static	96.86%	96.73%	95.83%	94.12%	90.14%
16-bit	97.28%	96.91%	95.84%	94.37%	90.24%
AdaptBit-HD	96.76%	96.68%	95.68%	93.63%	90.10%

B. Energy Efficiency, Execution Time, and Accuracy of AdaptBit-HD vs State-of-the-Art

Figure 7 shows the impact of AdaptBit-HD on accuracy and compares AdaptBit-HD with both a baseline of binary quantization and comparison with a 16-bits model. The graph clearly shows that AdaptBit-HD achieves comparable accuracy with 16-bit model. AdaptBit-HD loses only 0.1% accuracy when compared to a 16-bit models on average and AdaptBit-HD is 1% more accurate than binary models on average. Additionally, Table I compares the impact of dimensionality on accuracy across various bit representations in HD. As the results show, across all representations, accuracy does not drop significantly until reaching $D = 1,000$. This is expected, as prior work has shown that the higher dimensionality of HD primarily provides robustness [26]. Furthermore, the results show that simply reducing dimensionality by the same factor of reduction in bitwidth does not result in the same accuracy. Therefore, utilizing different quantization techniques like AdaptBit-HD are beneficial to gaining energy efficiency and faster execution time while maintaining higher accuracy.

Figure 8 compares the energy efficiency of AdaptBit-HD normalized to a 16-bit quantized model as a 16-bit model is able to achieve the same accuracy as full precision models with significantly less energy consumption. The figure also compares AdaptBit-HD with a binary quantized design which is the current state-of-the-art quantization for HD computing to achieve the best energy efficiency [9]. Figure 8 demonstrates that AdaptBit-HD is able to achieve energy efficiency close to that of the binary quantized model, where we define energy efficiency as the relative amount of overall energy consumption over the entire dataset. AdaptBit-HD is only 10% less energy efficient than the binary model. This is because AdaptBit-HD is able to terminate the hamming distance operation at the first bit the majority of the time, just like a binary model. This is demonstrated by the color coding of the stacked bar. The color coding of the different bits for AdaptBit-HD show the proportion of energy consumption spent on each bit. For example, the proportion of the bar that is colored blue is the proportion of energy spent on the first bit, which is approximately 90%. This is because, 90% of the time, only the first bit is used. This leads to approximately the same energy consumption as a binary model 90% of the time. Figure 9 additionally compares the execution time of AdaptBit-HD with a binary model and 16-bit model. The y-axis shows the speedup of AdaptBit-HD and binary models relative to a 16-bit model. As the graph demonstrates, we see a similar

comparison with execution time as energy efficiency. This is because the two are closely related and we accordingly see a similar speedup as well as energy efficiency improvement.

AdaptBit-HD is able to achieve 1.1% more accuracy than the binary model. Overall, AdaptBit-HD is comparable in energy efficiency and execution time to the binary model, but more accurate. Compared to the 16-bit model, AdaptBit-HD is $14.4\times$ more energy efficient at the cost of just 0.1% accuracy. This demonstrates that with an adaptive one bit at a time operation, AdaptBit-HD is able to achieve energy efficiency and execution time close the the binary model while maintaining accuracy comparable to full precision models. Therefore, AdaptBit-HD offers another point to the accuracy and energy efficiency trade-off curve that aims to suit application where energy efficiency and accuracy are more equally important. We can further see where AdaptBit-HD fits in the design space when we compare design with a 2-bit quantized model. A 2-bit quantized model achieves just 0.3% more accuracy than the binary model, while AdaptBit-HD is able to achieve 1.1% more accuracy than the binary model. Additionally, the 2-bit quantized model costs approximately 85% more energy consumption and is $1.81\times$ slower than the binary model. On the other hand, AdaptBit-HD is able to achieve just 10% more energy consumption and $1.07\times$ more execution time than binary models. This is because AdaptBit-HD is able to adaptively stop execution bit by bit for each sample, whereas the 2-bit quantization model needs to perform execution on $2\times$ the number of bits as the binary model for every sample. This makes AdaptBit-HD compelling for applications where both energy efficiency, execution time, and accuracy are equally important.

Additionally, by designing an FPGA accelerator for AdaptBit-HD, FPGAs can be reconfigured based on user application needs and changes. If a user needs highly efficient energy consumption and accuracy loss is not as important, they can use the binary model we compare with. For applications on the other side of the spectrum, they can use a higher bitwidth static model, e.g., a 5-bit model. Then, for applications that need to balance accuracy and energy efficiency such as running a medical diagnosis application, similar to our CARDIO dataset, on a mobile device with a battery, then AdaptBit-HD would be the best suited quantization method to use over both binary and higher bitwidth static quantization models.

C. AdaptBit-HD Area Comparison

At a glance, the most apparent drawback with our implementation of a bit-serial operation is that our design needs to store the full 5-bit class HVs all the time. This results in the area of our design being comparable to a 5-bit quantized model rather than the more space efficient binary model. Figure 10 shows the area comparison of a binary model with AdaptBit-HD relative to a 16-bit model. In this comparison we include the area of the FPGA resources as well as the area used for storing values on off chip DRAM. We do this because, comparing FPGA area only would not show much difference as, in FPGAs, static power consumption takes up a large proportion

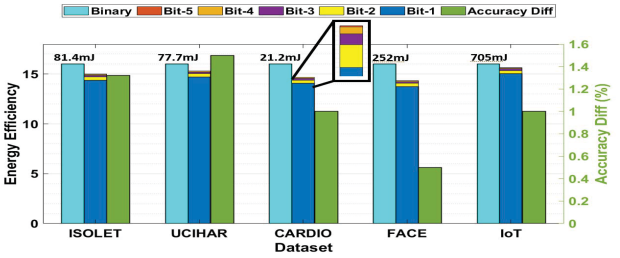

Fig. 8. Energy Breakdown of AdaptBit-HD and Comparison with Static Quantization Methods. Energy Efficiency is Shown Relative to a 16-Bit Model. Accuracy Difference is Compared to a Binary Model. The color coding of the different bits for AdaptBit-HD show the proportion of energy consumption spent on each bit.

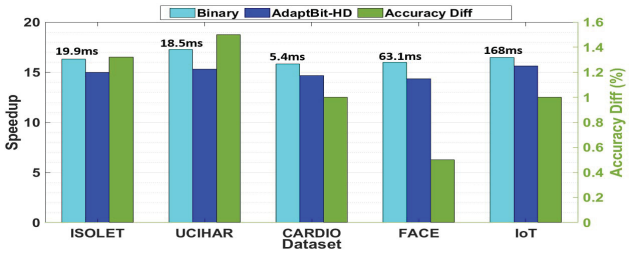

Fig. 9. Speedup of AdaptBit-HD and Comparison with Static Quantization Methods. Speedup is Shown Relative to a 16-Bit Model. Accuracy Difference is Compared to a Binary Model.

of the total power. Therefore, to maximize energy efficiency, designs replicate units until the FPGA is completely utilized to achieve the fastest execution time to reduce overall energy consumption. Therefore, almost all designs try to achieve high utilization of the FPGA. There are two comparisons on the figure, the first is during just inference and the other is an end-to-end implementation of HD, which includes encoding, training, retraining, and inference.

The figure shows, a binary design that performs inference only uses roughly 6% of the area of a 16-bit model as most of the area usage comes from storing the large class HVs. On the other hand, AdaptBit-HD uses close to 33% of the area of a 16-bit model. This is because AdaptBit-HD stores 5-bit HVs. However, if we look at an end-to-end implementation, the area usage converges. This is because both quantization methods need to store 16-bit precision class HVs during retraining and the entire training dataset in off chip DDR memory for retraining, which takes a bulk of the total area. Furthermore, we can see that AdaptBit-HD is able to achieve less area than a full 16-bit model end-to-end.

HD as a classification method is significantly more area efficient than other light-weight learning models such as SVMs. We compare with SVMs, because they offer similar accuracy to HD computing in most datasets and are also relatively light-weight compared to neural networks, just like HD computing. Here we show that SVMs use $110\times$ ($3.24\times$) more area during inference (end-to-end). This is because during inference, SVMs need to store all support vectors in their original data representation (32-bit). Similar to HD,

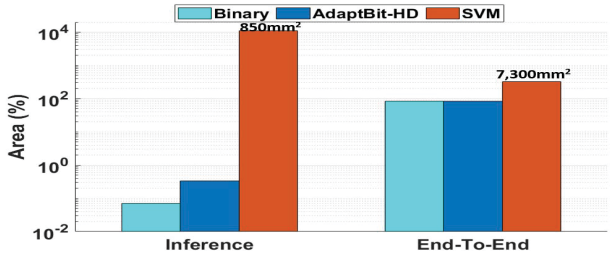

Fig. 10. Area Comparison of AdaptBit-HD, Static Quantization Methods for HD, and SVMs. Area Used is Shown Relative to a 16-bit Static HD Model. Including off-chip DDR for storage of data.

SVMs also need access to all training data during training. However, SVMs take $3.24\times$ more space due to HD mapping the data to high dimensional binary vectors rather than full precision feature vectors in off chip DDR.

V. Conclusion

This paper proposes AdaptBit-HD, an Adaptive Model Bitwidth Architecture for accelerating HD computing. AdaptBit-HD operates on the bits of the quantized model one bit at a time to save energy when fewer bits can be used to find the correct class. With AdaptBit-HD, we achieve both high accuracy by utilizing all the bits when necessary and high energy efficiency by terminating execution at lower bits when our design is confident in the output. Compared to binary quantization AdaptBit-HD is 1.1% more accurate at the cost of just 10% more energy consumption. Compared to a 16-bit static model, AdaptBit-HD is $14.4\times$ more energy efficient and $15.1\times$ faster at the cost of just 0.1% accuracy. This demonstrates that with an adaptive hamming distance operation, AdaptBit-HD is able to achieve energy efficiency and execution time close the the binary model while maintaining accuracy comparable to a 16-bit model.

Acknowledgements

This work was supported in part by CRISP, one of six centers in JUMP, an SRC program sponsored by DARPA, in part by SRC-Global Research Collaboration grant Task No. 2988.001, and also NSF grants 1527034, 1730158, 1826967, 1830331, 1911095, and 2003277.

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, "Encoding structure in boolean space," in *ICANN 98*, pp. 387–392, Springer, 1998.

[3] M. Imani, A. Rahimi, D. Kong, T. Rosing, and J. M. Rabaey, "Exploring hyperdimensional associative memory," in *High Performance Computer Architecture (HPCA), 2017 IEEE International Symposium on*, pp. 445–456, IEEE, 2017.

[4] P. Garcia Lopez, A. Montresor, D. Epema, A. Datta, T. Higashino, A. Iamnitchi, M. Barcellos, P. Felber, and E. Riviere, "Edge-centric computing: Vision and challenges," *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 5, pp. 37–42, 2015.

[5] M. Imani, J. Morris, S. Bosch, H. Shu, G. De Micheli, and T. Rosing, "Adapthd: Adaptive efficient training for brain-inspired hyperdimensional computing," in *2019 IEEE Biomedical Circuits and Systems Conference (BioCAS)*, pp. 1–4, IEEE, 2019.

[6] 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.

[7] Y. Kim *et al.*, "Efficient human activity recognition using hyperdimensional computing," in *IoT*, p. 38, ACM, 2018.

[8] M. Imani, S. Bosch, S. Datta, S. Ramakrishna, S. Salamat, J. M. Rabaey, and T. Rosing, "Quanthd: A quantization framework for hyperdimensional computing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2019.

[9] M. Imani *et al.*, "Bric: Locality-based encoding for energy-efficient brain-inspired hyperdimensional computing," in *ACM/IEEE Design Automation Conference (DAC)*, pp. 1–6, IEEE, 2019.

[10] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proceedings of the 44th Annual International Symposium on Computer Architecture*, pp. 1–12, 2017.

[11] M. Wess, S. M. P. Dinakarrao, and A. Jantsch, "Weighted quantization-regularization in dnns for weight memory minimization toward hw implementation," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2929–2939, 2018.

[12] S. Ye, T. Zhang, K. Zhang, J. Li, J. Xie, Y. Liang, S. Liu, X. Lin, and Y. Wang, "A unified framework of dnn weight pruning and weight clustering/quantization using admm," *arXiv preprint arXiv:1811.01907*, 2018.

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

[14] Y. Zhou, S.-M. Moosavi-Dezfooli, N.-M. Cheung, and P. Frossard, "Adaptive quantization for deep neural network," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[15] K. Wang, Z. Liu, Y. Lin, J. Lin, and S. Han, "Haq: Hardware-aware automated quantization with mixed precision," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 8612–8620, 2019.

[16] S. Jain, S. Venkataramani, V. Srinivasan, J. Choi, P. Chuang, and L. Chang, "Compensated-dnn: Energy efficient low-precision deep neural networks by compensating quantization errors," in *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*, pp. 1–6, IEEE, 2018.

[17] M. Imani, X. Yin, J. Messerly, S. Gupta, M. Niemier, X. S. Hu, and T. Rosing, "Searchd: A memory-centric hyperdimensional computing with stochastic training," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2019.

[18] M. Imani, D. Kong, A. Rahimi, and T. Rosing, "Voicehd: Hyperdimensional computing for efficient speech recognition," in *International Conference on Rebooting Computing (ICRC)*, pp. 1–6, IEEE, 2017.

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

[20] B. Philofsky, "Seven steps to an accurate worst-case power analysis using xilinx power estimator (xpe)," *White Paper: Spartan-3, Virtex-4, and Virtex-5 FPGAs WP353 (v1. 0) September*, vol. 30, 2008.

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

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

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

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

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

[26] J. Morris, K. Ergun, B. Khaleghi, M. Imani, B. Aksanli, and T. Rosing, "Hydrea: Towards more robust and efficient machine learning systems with hyperdimensional computing,"