Hierarchical, Distributed and Brain-Inspired Learning for Internet of Things Systems

Mohsen Imani[1*], Yeseong Kim[2*], Behnam Khaleghi[3], Justin Morris[4], Haleh Alimohamadi[5]
Farhad Imani[6], and Hugo Latapie[7]

[1]UC Irvine, [2]DGIST, [3]Qualcomm, [4]California State University San Marcos,
[5]UC Los Angeles, [6]University of Connecticut, and [7]CISCO

[*]Corresponding Authors: m.imani@uci.edu, yeseongkim@dgist.ac.kr

2023 IEEE 43rd International Conference on Distributed Computing Systems (ICDCS) | 979-8-3503-3986-4/23/$31.00 ©2023 IEEE | DOI: 10.1109/ICDCS57875.2023.00083

***Abstract*—In this paper, we propose EdgeHD, a hierarchy-aware learning solution that performs online training and inference in a highly distributed, cost-effective way. We use brain-inspired hyperdimensional (HD) computing as the key enabler. HD computing performs the computation tasks on a high-dimensional space to emulate functionalities of the human memory, such as inter-data relationship reasoning and information aggregation. EdgeHD exploits HD computing to effectively learn the classification models on individual devices and combine the models through the hierarchical IoT nodes without high communication costs. We also propose a hardware design that accelerates EdgeHD on low-power FPGA platforms. We evaluated EdgeHD for a wide range of real-world classification applications. The evaluation shows that EdgeHD provides highly efficient computation with reduced communication. For example, EdgeHD achieves on average 3.4× and 11.7× (1.9× and 7.8×) speedup and energy efficiency improvement during the training (inference) as compared to the centralized learning approach. It reduces the communication costs by 85% for the training and 78% for the inference.**

I. Introduction

Machine learning methods have been widely utilized to provide high quality for many cognitive tasks. Running sophisticated learning tasks requires high computational costs to process a large amount of learning data. A common solution is to use the cloud and data centers as the main central computing units. However, with the emergence of the Internet of Things (IoT), the centralized approach faces several scalability challenges towards high-performance computing [1], [2], [3], [4], [5], [6]. In IoT systems, a large number of embedded devices are deployed to collect data from the environment and produce information. The partial data need to be aggregated to perform the target learning task in the IoT networks such as a home- or even city-scale. It consequently leads to a high communication cost with high latency to transfer all data points to a centralized cloud.

Recent research work studied how to scale the learning tasks in a distributed fashion where the data are collected from different devices. A mainstream of the research is often referred to *federated learning* [7], [8], [9], [2]. For example, the study in [10] trains a central Deep Neural Networks (DNN) model over multiple devices where the data of each device have the same feature set.

However, effective learning in the IoT hierarchy is still an open question. We recognize the following technical challenges to scale the learning tasks for the IoT hierarchy. *(i) In reality, each IoT device has different types of sensors that generate heterogeneous features.* As an example, a smart home has many different edge nodes, e.g., a smart fridge, TV, stove, and personal devices. The contextual information of the smart home should be assimilated based on the collection of the sensor data. Distributing the learning tasks to the devices having heterogeneous data is not trivial for the federated learning solutions and existing algorithms, e.g., DNN and SVM (support vector machine). *(ii) The edge devices often do not have sufficient resources for online processing of the sophisticated learning algorithms*[11], [12]. The massive amount of data generated every day provides the opportunity to train a new model or at least update the pre-trained models. However, the existing learning algorithms are often over-complex to run on resource-constrained IoT devices. The state-of-the-art DNN acceleration solutions [13], [14] only process the inference tasks while assuming that the golden model can be trained in a centralized fashion. *(iii) To train and infer in a centralized fashion, the communication may dominate the total computing costs as the size of data generated in the swarm of the IoT devices increases.* Even if the learning tasks could be distributed to the edge devices by deploying a costly hardware accelerator, a large amount of data requires to be transferred between different nodes, e.g., inputs and outputs of neurons for DNN models, during the model training procedure. In addition, reliable communications are not granted, and IoT networks are often deployed assuming harsh network conditions [15].

In this work, we seek to enable distributed learning using the data that heterogeneous sensors for each IoT device generate on the fly. We accelerate the learning tasks by utilizing the IoT devices as *federated* computing units, i.e., the learning tasks are processed on the local embedded devices located in the hierarchy. Many IoT devices are capable of processing a part of computational tasks with limited resources, e.g., smart gateways [16], [17]. To provide an effective and lightweight learning solution for the IoT hierarchy, we use brain-inspired hyperdimensional (HD) computing as the machine learning solution [18], [19]. HD computing is an alternative computing approach for cognitive tasks. HD computing mimics crucial properties of the human memory using high-dimensional vectors, called *hypervectors*. For example, the brain efficiently

2575-8411/23/$31.00 ©2023 IEEE
DOI 10.1109/ICDCS57875.2023.00083

aggregates and understands the relationship between data. In HD computing, adding hypervectors imitates data aggregation, and we can quantify the inter-data relationship based on the hypervector similarity. In recent years, HD computing has been employed in a range of applications as a lightweight machine learning for activity recognition [20], biomedical signal processing [21], [**?**], genomics [22], reinfocement leanring [23], [24], [25], and reasoning [26]. A key HD computing advantage is its robust and efficient training capability in one or few shots where object categories are learned from a few examples and in a few iterations instead of many samples and iterations [19]. HDC has achieved comparable to higher accuracy compared to support vector machines (SVMs) [27], [28], gradient boosting [29], and convolutional neural networks (CNNs) [30] with much lower execution energy. Recently, several companies started exploiting the HD computing capability to enable intelligence in IoT devices [31], [32].

HD computing has many attractive properties that address the aforementioned challenges, making it desirable for distributed learning in the hierarchy. HD computing can be performed with linear combinations of hypervectors which non-linearly map the raw data to an HD space. Thus, *we can easily aggregate the trained models and the sensor data* produced by different sensor nodes in the hierarchy. In addition, the learning procedure is performed using well-defined hypervector operations without overcomplex learning steps such as backpropagation in neural networks. *The hypervector operations can be efficiently performed on inexpensive, low-power platforms*, e.g., FPGA, since most computations are (dimension-)independent. The less-powerful IoT devices can thus perform cognitive tasks efficiently with minimal deployment costs. Besides, *a hypervector can combine multiple information in a space-effective way,* hence we can significantly reduce the communication costs when running the computations over a collection of multiple devices. The trained model is extremely robust in the possible presence of hardware/network failures which is common in IoT systems.

In this paper, we propose a novel hierarchy-aware brain-inspired learning, called EdgeHD, which enables online training and inference on edge devices with significantly high computation efficiency. Our key contributions are:

- **EdgeHD effectively distributes the learning tasks into edge devices that produce heterogeneous data in a hierarchy in an IoT system.** EdgeHD is capable of updating the learning model online through the hierarchy to provide better prediction results based on users' feedbacks. During the inference, it automatically decides where to run the tasks to guarantee the desired prediction quality.
- **Our approach exploits the mathematical properties that govern the high-dimensional space to compress and transfer data through the hierarchy.** This approach reduces the computation as well as data communication cost by eliminating the necessity of moving all data between nodes.
- **We show an efficient hardware implementation based on low-power FPGA.** Our design supports online training and inference in a fully pipelined structure. In the hierarchical setting, the FPGA design located in each node performs the learning tasks with a minimal power consumption of 0.28W on average, while the state-of-the-art DNN accelerators, e.g., Google Tensor Processing Unit (TPU) consume at least 290W [33].
- **EdgeHD significantly improves the classification accuracy of the existing HD computing algorithms.** In contrast to the existing methods, which linearly map each input feature into the hyperspace, the proposed solution explicitly considers non-linear interactions between the inputs. We exploit Radial Basis Function (RBF) kernel [34], [35] to design a highly accurate classifier. Our evaluation shows that the proposed approach achieves comparable accuracy as sophisticated learning algorithms such as SVM and DNN, while providing significantly higher efficiency than prior HD computing-based classifiers.

We evaluate EdgeHD on practical IoT workloads consisting of multiple sensing nodes with different network hierarchies. Our evaluation shows that the proposed hierarchical training provides comparable accuracy to the state-of-the-art method while achieving 3.4× and 11.7× (1.9× and 7.8×) speedup and energy efficiency as compared to a state-of-the-art centralized learning approach during the training (inference). One of the primary sources of the improvements is the significant reduction in the communication costs, e.g., 85% for the training and 78% for the inference.

II. Motivational Scenario

We consider a smart home as an example of an IoT system, where different appliances generate data to perform the desired learning task, e.g., activity recognition for household members or power usage management on a city scale (Figure 1a). The data from each device is aggregated on the gateways at the house level. Finally, the data of all gateways are aggregated in a server, i.e., a central node. By analyzing the data, we can develop a learning model which can be used at different levels, such as appliances, houses, or even a city.

Figure 1b shows a centralized approach which is typically used in systems nowadays. This approach aggregates all the generated data to a server located at the city level. Then, the cloud creates a unified model which can be used in all other nodes. This approach has several disadvantages. First, transferring a large amount of data to the cloud would be slow and unstable due to the nature of the wireless networks commonly used by smart devices in home-scale systems. In addition, it does not guarantee online training for a real-time response to inference tasks with updated models.

In this paper, we propose a hierarchy-aware learning approach that enables distributed learning where each edge device performs its learning task (Figure 1c). Instead of transferring all data points to the cloud, each edge device trains its own model using the portion of data available at that particular node. Next, instead of the raw data, the *models* are aggregated through the hierarchy, such that a node at the house level gathers information from all connected end-node devices. Finally, all the house models are aggregated at the city level to create

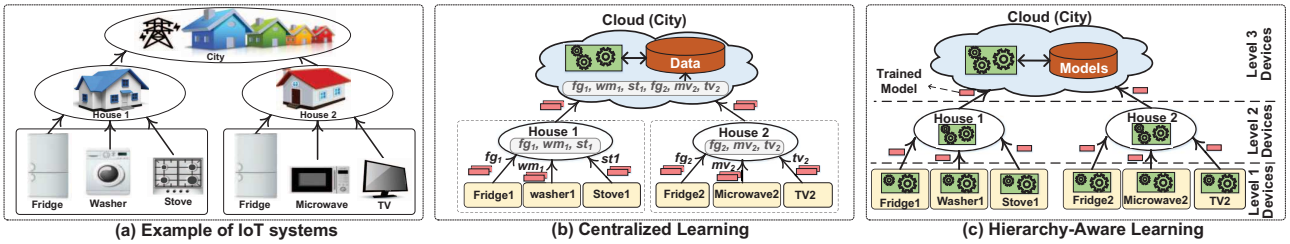


Fig. 1. (a) Smart home as an example of IoT system, (b) centralized training and (c) hierarchy-aware learning.

a model based on all features generated by the end node devices. It should be noted that, in practice, recent appliances have already been produced with computing capability on the edge, e.g., using ARM-based processors of smart fridges and televisions.

The major technical challenge is how to aggregate the models learned on each device in a distributed manner. For example, the learned models of embedded devices such as a fridge, TV, and stove need to be aggregated on the smart gateway located on the house level and beyond. In existing learning algorithms, e.g., DNN and SVM, this aggregation is not trivial.

Why Hyperdimensional Computing? In this work, we tackle this problem by exploiting brain-inspired HD computing. The brain's circuits have many neurons and synapses, suggesting that large circuits are fundamental to mimicking brain functionalities. HD computing imitates human memory using ultra-wide words, i.e., hypervectors, which have tens of dimensions [18]. Once the original data are encoded to hypervectors, various brain functionalities can be modeled with hypervector operations.

The proposed EdgeHD exploits the following properties of HD computing to enable hierarchical learning. (i) The key operation of hierarchical learning is data aggregation. HD computing represents the effective information aggregation of the human memory model using simple linear combinations of hypervectors such as addition and multiplication. This allows us to propagate the learning models/data through the hierarchy without complex computations. (ii) As human memory does, HD computing can also perform continuous learning by adding more information into the hypervector model, which is expected in online learning. (iii) We exploit the fact that a hypervector can store multiple pieces of information to reduce communication overheads. The bio-inspired hypervectors store information with i.i.d random components. That is, every component has the same responsibility to represent a datum, making HD extremely robust against most failure mechanisms and noises, which is common in IoT systems. (iv) Despite the high dimensionality, HD computing can be implemented in a resource-effective manner since the hypervectors often are represented with low-precision components, e.g., binaries, and highly parallelizable as most operations in the hyperspace are dimension-independent.

III. HD Computing Classification

Figure 2a shows the overview of the proposed classification method using HD computing. For the sake of simplicity, we illustrate the algorithm without explicitly regarding the device hierarchy; we will revisit this algorithm in Section IV to enable distributed learning. In the initial training step, HD computing performs the learning task after mapping all training data into the high-dimensional space. The mapping procedure is often referred to as *encoding*. Ideally, the encoded data should satisfy the common-sense principle: data points that are different from each other in the original space should also be different in the high-dimensional space.

To find the universal property for each class in the training dataset, we linearly combine hypervectors belonging to each class, i.e., adding the hypervectors to create a single hypervector for each class. Once combining all hypervectors, we treat per-class accumulated hypervectors, called *class hypervectors*, as the learned model. Next, a similarity search procedure performs the inference task. For a given *query* hypervector encoded for a tested data point, it selects the class with the most similar hypervector.

A. Non-Linear Encoding

There are multiple encoding methods proposed in literature [36], [37], [38]. Although these methods have shown excellent quality of learning, these techniques are applications specific. We introduce a universal hyperdimensional encoding process that prepares encoded data for both learning and cognition. Suppose that the input of the encoder is a 2D image $\mathbf{F}$ with size $n \times n$. The pixel at location (X, Y) is defined as $\mathbf{F}_{X,Y}$. For image dimensions x and y, we randomly generate two base hypervectors $\mathbf{B}_x$ and $\mathbf{B}_y$ with dimensionality $D \gg n$. They are defined as follows: $\mathbf{B}_x = e^{i\vec{\theta}_x/w_x}$ and $\vec{B}_y = e^{i\vec{\theta}_y/w_y}$, where $\theta \in \{\mathcal{N}(0,1)\}^D$ and w is the length scale. To represent a certain location on an axis, we attach the index to the power of the exponential such as $\mathbf{B}_x^{X_1}$ and $\mathbf{B}_y^{Y_1}$. Here, we take $\mathbf{B}_x$ as an example to define the similarity metric on the x-axis of the image: $\delta(\mathbf{B}_x^{X_1}, \mathbf{B}_x^{X_2}) \overset{D\to\infty}{\approx} k(\frac{X_1 - X_2}{w_x})$ where $k(\cdot)$ is the standard Gaussian kernel; X_1 and X_2 are two locations on the x-axis. The same similarity metric is also defined on the y-axis. In more than $1D$, the kernel becomes $k_{gen}(\mathbf{F}_1 - \mathbf{F}_2) = k(||\mathbf{F}_1 - \mathbf{F}_2||)$

Using base hypervectors, we generate an index hypervector or ID hypervector for each pixel (X, Y) using element-wise multiplication: $\mathbf{B}_x^X * \mathbf{B}_y^Y$. The Gaussian kernel ensures that the identification (ID) hypervectors are correlated between nearby pixels, and thus helps maintain spatial information during the encoding. To represent each pixel in hyperspace, we multiply the value of pixel $F_{X,Y}$ with its ID hypervector $\mathbf{B}_x^X * \mathbf{B}_y^Y$.

With components defined above, we encode the image f to its corresponding hypervector $\mathbf{V}_F$ using the element-wise sum of multiple pixel hypervectors, also known as the bundling operation in HDC:

$$\mathbf{V}_F = \sum_{X,Y} P_{X,Y} \mathbf{B}_x^X * \mathbf{B}_y^Y$$

For a large but finite dimensional mapping Z, the shift-invariant kernel K as the one defined above can be approximated using inner-products [39]:

$$K(\mathbf{F}_1 - \mathbf{F}_2) \approx \mathbf{H}_D(\mathbf{F}_1)^T \mathbf{H}_D(\mathbf{F}_2) \quad (1)$$

where D is the dimensionality of the mapping. There are several practical measures to design the mapping $\mathbf{H}$ that corresponds to known kernels. In this paper, we focus on one of them that approximates the RBF kernel, and it is defined as follows:

$$\mathbf{H}_D(\mathbf{F}) = \sqrt{\frac{2}{D}} cos(\mathbf{B} \cdot \mathbf{F} + \mathbf{b}) \quad (2)$$

$\mathbf{B}$ is a vector of dimension D with its elements randomly sampled from standard Gaussian distribution $\mathcal{N}(0,1)$ and $\vec{b}$ functions as a bias vector with elements sampled from the uniform distribution $\mathcal{U}(0, 2\pi)$. Once they are randomly generated, we keep them fixed during the later learning and inference.

Figure 2b shows our encoding procedure. Let us consider an encoding function that maps a feature vector $\mathbf{F} = \{f_1, f_2, \ldots, f_n\}$, with n features ($f_i \in N$) to a hypervector $\mathbf{H} = \{h_1, h_2, \ldots, h_D\}$ with D dimensions ($h_i \in \{-1, 1\}$). We generate each dimension of the encoded data by calculating a dot product of the feature vector with a randomly generated vector as $h_i = cos(\vec{B}_i \cdot \vec{F} + b) \times sin(\vec{B}_i \cdot \vec{F})$, where B_i is the randomly generated vector with a Gaussian distribution (mean $\mu = 0$ and standard deviation $\sigma = 1$) with the same dimensionality of the feature vector and b is a random value sampled uniformly from $[0, 2\pi]$.

The random vectors $\{\mathbf{B}_1, \mathbf{B}_2, \cdots, \mathbf{B}_D\}$ can be generated once offline and then can be used for the rest of the classification task. After this step, each element h_i of a hypervector $\mathbf{H}^n$ has a non-binary value. In HD computing, binary (bipolar) hypervectors are often used for computation efficiency. We thus obtain the final encoded hypervector by binarizing it with a sign function.

B. Classification Learning and Inference

Training: In the training step, we combine all the encoded hypervectors of each class using the element-wise addition. For example, in an activity recognition application, the training procedure adds all hypervectors which have the "walking" and "sitting" tags into two different hypervectors. Where $\mathbf{H}_j^i = \langle h_D, \cdots, h_1 \rangle$ is encoded for the j^{th} sample in i^{th} class, each class hypervector is trained as follows:

$$\mathbf{C}^i = \sum_j \mathbf{H}_j^i = \langle c_D^i, \cdots, c_1^i \rangle$$

The element-wise addition results in generating non-binarized class hypervectors, i.e., $c_d^i \in N$. Note that since our encoding method projects the original data non-linearly to the high dimensional space, the linearly combined model can perform well even on non-linearly separable data.

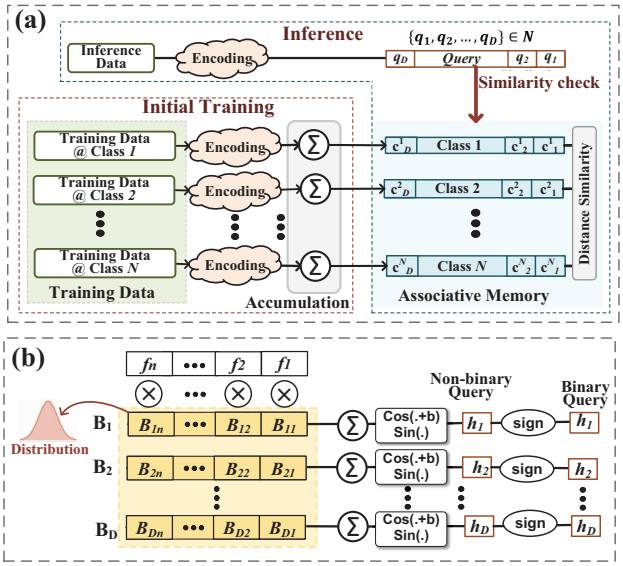


Fig. 2. Overview of HD computing for classification.

Once the initial training is done, we train the class hypervector model again to improve the classification accuracy. In this *retraining* step, we calculate the similarity between each encoded hypervector and trained model to check whether the data sample is correctly classified or not. If the encoded hypervector, $\mathbf{H}$, is correctly classified by the current model, we will make no changes to the model. Otherwise, we update the model by respectively adding and subtracting it from the correct and incorrect classes as follows:

$$\overline{\mathbf{C}}^{correct} = \mathbf{C}^{correct} + \mathbf{H} \quad and \quad \overline{\mathbf{C}}^{wrong} = \mathbf{C}^{wrong} - \mathbf{H}$$

The retrained model provides a better fit to the training data and gets higher accuracy. We repeat the same procedure for multiple iterations. In our observation, repeating 20 iterations yields sufficient convergence for all the tested datasets.

Inference (testing): The main computation of the inference is the encoding and associative search. We perform the same encoding procedure to convert a test data point into a hypervector, called *query hypervector*, $\mathbf{Q} \in \{0, 1\}^D$. Then, it computes the similarity of the query hypervector with all k class hypervectors, $\{\mathbf{C}_1, \mathbf{C}_2, \cdots, \mathbf{C}_k\}$. We measure the similarity between a query and a i^{th} class hypervector using: $\delta\langle \mathbf{Q}, \mathbf{C}_i \rangle$, where δ denotes the similarity metric. After computing all similarities, each query is assigned to a class with the highest similarity.

The proposed HD-based classification algorithm is appropriate for hierarchical and distributed learning since the training and inference computations can be decomposed with simple linear combinations of hypervector operations where the hypervectors are already encoded in a non-linear manner. There are two remaining technical questions: i) how to combine multiple hypervectors provided by multiple children nodes through the hierarchy and ii) how to reduce the cost of communications to transfer hypervectors between the nodes.

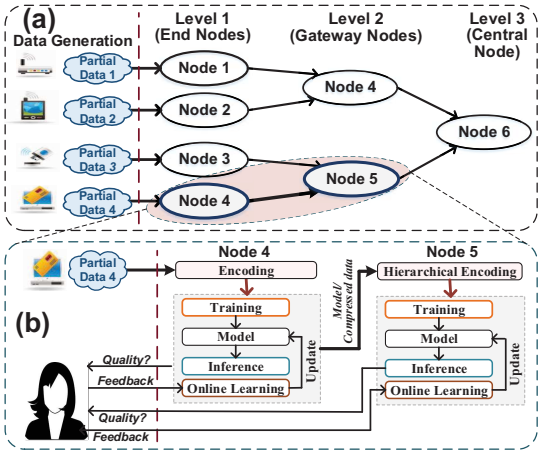


Fig. 3. Overview of Hierarchical Hyperdimensional Computing Framework (EdgeHD)

In the next section, we describe how the HD classification algorithm can be effectively applied to the IoT hierarchy by addressing these issues.

IV. Hierarchical Learning

Figure 3a shows an overview of EdgeHD, which performs the HD computing on each device in a distributed manner, where multiple computing nodes are connected in a hierarchical network. The *end node* devices which are located in the first level of hierarchy, run the original encoding and training procedure with collected data points, creating a partial model based on the available data on that node. The partial models are sent to the *gateway* nodes in the upper layers to learn different models using the data provided by children nodes. The learning task is propagated to the *central* node at the highest level in a hierarchical fashion.

Figure 3b shows the classification tasks performed on each node. While the end node runs the encoding procedure described in Section III-A, the gateway and central nodes carry out a *hierarchical encoding* procedure to aggregate the partial models or hypervectors (Section IV-A.) Since the hypervector created by the hierarchical encoding procedure includes the information collected on the children nodes, EdgeHD can train more comprehensive models on higher-level nodes (Section IV-B.) During the inference step, users can use any models stored through the hierarchy (Section IV-C.) For example, if the user wants to have a real-time response, the model of the end node would be preferable; if a higher quality of prediction is more desired, they can use the model of the central node at the expense of the communication costs to consider the collected data of other nodes.

In the hierarchical learning mode, EdgeHD also supports online training, which utilizes users' feedback to update the models trained offline (Section IV-D.) During the runtime, it is usually hard to expect labeled data, i.e., correct classes for the observed samples. Considering the circumstances, we show how to update models when users are willing to provide only negative feedback, i.e., when users are unsatisfied with the classification results.

A. Hierarchical Encoding

The encoding procedure described in Section III-A maps the feature values in the original space to the hyperspace. In contrast, the hierarchical encoding procedure aggregates multiple *hypervectors* to create a single hypervector. Since the node at the lower level has a less number of features, it typically needs a smaller dimension to keep the information accessible on the node. Thus, we set different hypervector dimensionalities for each node. Let us assume that a large enough dimension, e.g., D=10,000, can include the information of all data points of n features separately collected in the hierarchy. Where the node at the top of the hierarchy has the highest dimensionality, i.e., D, EdgeHD assigns the dimensions of the devices according to their available feature sizes. For instance, if a device is connected to the end nodes in the hierarchy and the end nodes in total collect n_i features, the device gets a dimensionality of $d_i = D \times n_i/n$.

With the determined dimensionality for each node, EdgeHD performs the encoding procedure for any hypervectors in a hierarchical manner. Figure 4 shows the proposed hierarchical encoding with an example of three nodes. Each end node in the first level (Node 1 and Node 2) uses the original encoding scheme (Section III-A) and produces a hypervector which usually has a relatively small dimensionality (d_1 and d_2). Then, a gateway node (Node 3) first creates a hypervector by concatenating the multiple hypervectors provided by the children nodes. Although this simple concatenation yields the hypervector of the desired dimensionality, $d_1 + d_2$, it does not consider possible interactions between different features of the children nodes.

To address this issue, we exploit projection matrix elements randomly selected from $\{-1, 0, 1\}$. EdgeHD performs a vector-to-matrix multiplication for the concatenated hypervector and the projection matrix. The results of the multiplications are binarized again with the $sign()$ function. It maps the concatenated hypervector into another space while randomly combining the elements of dimensions of the input hypervectors, and thus the projected hypervector considers the interactions of the input hypervectors concatenated before. The projected hypervector has *holographic* distribution [18], meaning that all feature values have the same impact on generating each dimension of the hypervector. The holographic distribution enables high robustness of HD computing even when using either uncertain networks or unreliable hardware commonly seen in IoT systems, which may lead to loss of some dimension values. In Section VI-F, we discuss the detailed evaluation of holographic encoding.

B. Model Training in Hierarchy

The first step of the training is to learn the initial class hypervectors on each device. As discussed in Section III-B, the initial model is generated by the addition of all the encoded hypervectors. Each device only needs to transfer its HD model, i.e., k class hypervectors, to its parent node rather than sending all the encoded hypervectors for the data points. The gateway and central nodes can train their model by

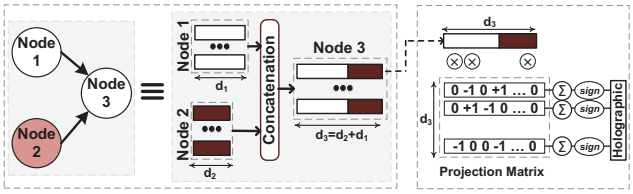


Fig. 4. Model/data aggregation on hierarchical nodes: (a) concatenation, (b) random projection of concatenated model/value for holographic distribution.

performing the hierarchical encoding procedure for the received class hypervectors. This approach enables the initial training to perform in a distributed way on all nodes while significantly reducing communication costs.

To achieve higher classification accuracy, we also perform the retraining step on each device. A naive implementation is to propagate the encoded hypervector for each sample through the hierarchy. However, communication costs would be high in this case. We address this issue by sending the hypervectors with multiple batches. Let us assume that $\mathbf{H}^i$ is a set of hypervectors for the i^{th} class encoded from the training dataset on an end node. For a given batch size, B, i.e., the number of hypervectors combined in a batch, EdgeHD splits $\mathbf{H}^i$ into multiple subsets that have the size of B, and performs the element-wide addition for each subset, creating $\beta^i = \lceil |\mathbf{H}^i| / B \rceil$ *batch hypervectors*. In this case, each node only sends $\sum_i^K \beta^i$ hypervectors where K is the number of classes. The gateway and central nodes run the hierarchical encoding procedure to combine the batch hypervectors provided by the children, and the retraining procedure described in Section III-B can be applied for each batch.

There is a tradeoff between batch size and accuracy. For example, suppose it does not use the batch method, i.e., $B = 1$, since the nodes in the higher levels can check the similarity for each sample. In that case, we may achieve higher accuracy at the expense of the communication cost.

C. Hierarchical Inference

The inference is performed by checking the similarity of a query hypervector with all k class hypervectors to assign a class with the highest similarity as explained in Section III-B. In EdgeHD, the inference can be carried out on any model located in different nodes. For example, we may perform a local inference using the model stored in each end node. When high prediction quality is required, we may send the encoded hypervector up to the central node. EdgeHD exploits the following two techniques to reduce the communication costs to send the query hypervectors.

Automated Decision on Inference Node: EdgeHD decides where to perform the inference task by estimating the confidence level for each prediction. The confidence level depends on the similarity distances to the class hypervectors. For example, if the similarity between a query hypervector and the matched class hypervector is relatively higher than others, we can consider it a trustworthy prediction. We compute the relative similarity using the softmax function, whose inputs are the normalized cosine similarity values to the class query

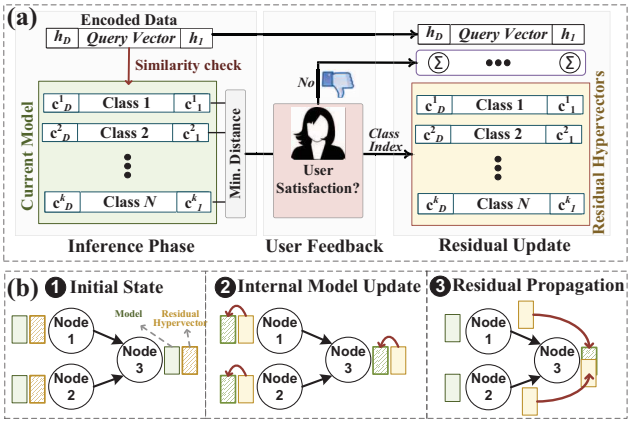


Fig. 5. EdgeHD framework supporting online learning based on the user's feedback during the inference.

hypervectors for all k classes. A larger value indicates a higher difference between the cosine similarity values, meaning that the prediction result is highly confident. Thus, EdgeHD compares the confidence level of the matched class with a user-configurable threshold value. If the prediction results are not sufficiently confident, the device sends the hypervectors to the higher nodes so that the higher-level device can perform the inference by considering the data provided by other nodes.

Hypervector Compression: The second technique is to compress multiple hypervectors into a single hypervector. Our compression method exploits the mathematical orthogonality of random vectors in high-dimensional space. Let us assume m hypervectors $\{\mathbf{H}_1, \mathbf{H}_2 \ldots, \mathbf{H}_m\}$ to be compressed. The batching procedure utilizes m random bipolar hypervectors, called *position* hypervectors, $\{\mathbf{P}_1, \mathbf{P}_2, \ldots, \mathbf{P}_m\}$ with the same dimensionality ($\mathbf{P}_i \in \{-1, 1\}^D$). Since the position hypervectors are generated randomly, they will have a nearly orthogonal distribution [40]:

$$\delta\langle \mathbf{P}_i,\ \mathbf{P}_j\rangle \simeq 0 \qquad (0 < i, j \leq m,\quad i \neq j)$$

With the randomly generated hypervectors, the compression procedure is done as follows:

$$\mathbf{H} = \mathbf{P}_1 * \mathbf{H}_1 + \mathbf{P}_2 * \mathbf{H}_2 + \cdots + \mathbf{P}_m * \mathbf{H}_m \tag{3}$$

The combined $\mathbf{H}$ hypervector can store the information of all $\mathbf{H}_i$ hypervectors as well as the order of combinations. To extract the i^{th} hypervector, we can multiply the compressed hypervector with the corresponding position hypervector:

$$\mathbf{H_i} \approx (\mathbf{H})\cdot\mathbf{P'}_i = \underbrace{\mathbf{H_i} * (\mathbf{P}_i \cdot \mathbf{P}_i)}_{Signal} + \underbrace{\sum_{j,\forall j \neq i} \mathbf{H_j} * (\mathbf{P}_i \cdot \mathbf{P}_j)}_{Noise} \tag{4}$$

In this equation, since $\mathbf{P}_i \cdot \mathbf{P}_i$ is the hypervector whose elements are all zeros, the signal term is equal to $\mathbf{H_i}$. Besides, due to the near orthogonality between different position hypervectors, the noise term is very close to zero. Compressing more hypervectors increases the amount of noise.

D. Online Model Updating

During runtime, users might not satisfy with the classification quality that the models in any particular nodes provide. We

propose the idea of online training, which updates the models in the hierarchical setting. Figure 5 shows the workflow of the online learning procedure. The model updates happen based on the user's feedback for the inference results. Since labeling every observation would not be feasible in reality, we assume that users would be willing to provide feedback when they are not satisfied.

Since the negative feedback means that the prediction is made by an incorrect class hypervector, EdgeHD subtracts the query hypervector from the class hypervector currently selected. We may perform it for every negative feedback; however, this approach has two main drawbacks: first, it may degrade the inference efficiency since updating the model delays the processing of the next inference task. Second, it results in high communication costs to propagate the hypervectors through the hierarchy.

EdgeHD updates the models less frequently by accumulating the hypervectors for the negative feedback. Figure 5a shows the online learning procedure for each node. Each device maintains K hypervectors whose elements are initially assigned to zero, called *residual hypervectors*. Each residual hypervector corresponds to one of the classes. Once negative feedback is given, we add the query hypervector to a residual hypervector corresponding to the class with an incorrect match. Each edge device continuously performs the inference while accumulating to the residual model.

Figure 5b shows how EdgeHD updates the models through the hierarchy. In the initial state, each device has its own residual hypervectors (❶). Each device first updates its own model by subtracting the residual hypervectors from the current model (❷). Then, the residual hypervectors are propagated to the parent nodes (❸), so that the gateway and central nodes run the hierarchical encoding to combine the different residual hypervectors and update the model. The online update procedure can be initiated at anytime, depending on the users' requirements. For example, in a smart home, users may want to update the model every night.

V. Hardware Acceleration

EdgeHD can be accelerated in different platforms. Since there is no dependency between dimensions during the hypervector operations, FPGA is suitable hardware to parallelize the HD computations in a power-efficient way. Figure 6 shows the overview of our FPGA design. In the followings, we explain the details of the implementation.

A. Encoding

EdgeHD encodes each data point by computing the inner product of a feature vector with different weight vectors. Since the Gaussian distribution creates many near-zero values, we can easily create *sparse* random vectors to reduce the number of multiplications. The weight vectors can be stored with a vector with $(1-s)\times n$ consecutive non-zero values, and an index value that represents the index of the first non-zero element where s is the sparsity factor and n is the number of features. This index can be stored as $\log_2 n$ bits. All weight vectors are stored in Block RAM (BRAM), which is on-chip FPGA memory

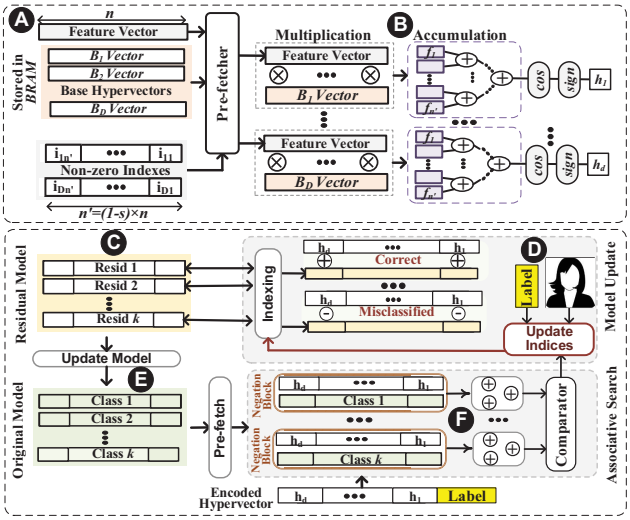


Fig. 6. EdgeHD FPGA implementation.

(Figure 6Ⓐ). During the encoding, our approach prefetches the weight vectors from the BRAM blocks and stores them in the locally distributed memory that can be accessed faster than BRAM. During the encoding, FPGA reads the first m features of original data points ($m \leq n$). Next, it accesses the weight vector and then multiplies $(1-s)\times n$ continuous dimensions of the feature vector with the corresponding weight vector. These multiplications are processed using Digital Signal Processor (DSP) blocks, and they are parallelized for different weight vectors. The results of the inner products are accumulated using a tree-based adder structure (Figure 6Ⓑ). Finally, the cosine function is calculated using the lookup table (LUT) logic. Finally, the encoded hypervector can be binarized by considering the sign of the encoded data as a binary output.

B. Training & Inference

EdgeHD has three types of learning procedures that update the class hypervector, i.e., the model: i) initial training, ii) retraining, and iii) online learning. Since they perform a very similar computation, we design a unified FPGA implementation. Instead of directly changing the class hypervector stored in BRAM, our hardware design accumulates all required changes for the class hypervectors within the residual hypervectors, which store a vector corresponding to each class (Figure 6Ⓒ). For example, in the initial training and online learning, only the element-wise addition is performed, whereas the subtraction is required additionally for the retraining step (Figure 6Ⓓ) depending on the results of the associative search. Finally, EdgeHD updates the original model with the residual hypervectors once (Figure 6Ⓔ).

The associative search, i.e., computing the similarity metric, is a common operation used in both the retraining and inference (Figure 6Ⓕ.) To reduce the cost of cosine similarity, we devise two optimization techniques. First, we simplify the cosine similarity to the dot product by pre-normalizing the class hypervectors once after each training step. We also ignore the normalization of each query hypervector, as this vector is a common term for all classes. Second, we eliminate the

TABLE I
DATASETS (n: FEATURE SIZE, K: NUMBER OF CLASSES)

	n	K	# End Nodes	Train Size	Test Size	Description
MNIST	784	10	NA	60,000	10,000	Handwritten Recognition[41], [42]
ISOLET	617	26	NA	6,238	1,559	Voice Recognition [43]
UCIHAR	561	12	NA	6,213	1,554	Activity Recognition(Mobile)[44]
EXTRA	225	4	NA	146,869	16,343	Smartphone Context Recognition[45]
FACE	608	2	NA	522,441	2,494	Face Recognition[46]
PECAN	312	3	312	22,290	5,574	Urban Electricity Prediction [47]
PAMAP2	75	5	3	611,142	101,582	Activity Recognition(IMU) [48]
APRI	36	2	3	67,017	1,241	Performance Identification[49]
PDP	60	2	5	17,385	7,334	Power Demand Prediction [50]

multiplication involves between the query and class hypervector since EdgeHD works with binary query vectors. We prefetch each class hypervector to the *negation* block and flip the sign bit of each element depending on the query bits. The results of the negation block are then transferred to a tree-based adder for accumulation. Finally, a comparator located at the right-hand side of the associative search finds the class with the highest similarity (Figure 6❼).

VI. EVALUATION

A. Experimental Setup

We implement an in-hour simulation framework based on NS-3 [51] to evaluate how EdgeHD performs on a large-scale IoT hierarchy. The simulation framework evaluates EdgeHD in a *hardware-in-the-loop* fashion. We use NS-3 to simulate communications on hierarchical network topologies with diverse network mediums. During the simulation loop, the simulator invokes the EdgeHD learning procedures (wrapped with ApplicationContainer of NS-3) on actual platforms which represent different nodes in the IoT hierarchy. We implement EdgeHD on Raspberry Pi (RPi) 3B+ as a model of the end nodes and gateway nodes. For FPGA, we design the EdgeHD functionality using Verilog and synthesize it using Xilinx Vivado Design Suite [52]. The synthesis code has been implemented on the Kintex-7 FPGA KC705 Evaluation Kit. For the central node, we developed a CUDA-based implementation of the proposed technique on a server system, which uses Intel Core i7-8700K CPU and NVIDIA GPU GTX 1080 Ti. The simulator collects the execution time and measures the power consumption for each connected platform while running the learning procedures. The power consumption is collected by Hioki 3337 power meter.

Table I summarizes the evaluated datasets. The tested benchmarks consist of canonical classification datasets such as voice recognition, smartphone context recognition, and a large dataset for face recognition which includes hundreds of thousands of images. For hierarchy-aware, we use four datasets. (i) **PECAN** presents a dense urban area where a neighborhood may have hundreds of housing units [47]. It has 52 houses observed, where a set of appliances instrumented with sensors records average energy consumption. The goal is to predict the level of power consumption in urban area. (ii) **PAMAP2** (Physical activity monitoring) is a dataset for human activity recognition which is widely used to understand user contexts [48]. The data are collected by four sensors (three accelerometers and one heartbeat sensor), producing 75 features in total. (iii) **APRI** (Application performance identification) is collected on a small server cluster that consists of three machines [49]. The server cluster runs Apache Spark applications while collecting performance monitoring counter (PMC) events on each server. The goal is to identify two workload groups depending on their computation intensity. (iv) **PDP** (Power demand prediction) is collected on another high-performance computing cluster consisting of six servers [50]. The goal is to identify either the high or low power state of a server using PMC measurements of the other five servers in the cluster.

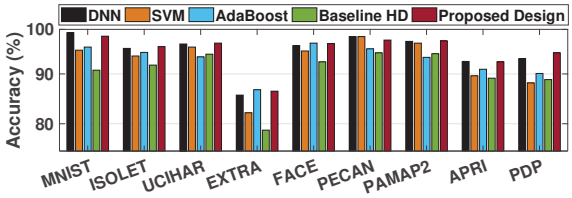


Fig. 7. Classification accuracy comparison.

We evaluated two hierarchical network topologies: (i) a star structure (STAR) that all end-node devices are directly connected to the central node without having gateway nodes, and (ii) a three-level tree structure (TREE) with the gateway nodes which have two end nodes as the children and the central node as the parent. For example, since the APRI has five end nodes, we use two gateways to aggregate the models/data. The central node connects two gateways, and one end node remains. EdgeHD has configurable parameters that users can set according to their efficiency requirements. We use the following parameters if not noted. (i) dimension size (D): 4,000, (ii) the batch size in the model updates (B): 75, (iii) the compression rate during the inference (m): 25, and (iv) the confidence threshold is 0.75. We also show an in-depth exploration of the parameters in the experimental results.

B. Classification Accuracy Comparison

HD computing vs. state-of-the-art: Figure 7 compares the classification accuracy with the state-of-the-art classification algorithms, including Deep Neural Network (DNN), Support Vector Machine (SVM), and AdaBoost. We also compare the accuracy of EdgeHD with a state-of-the-art HD-based classifier published in [36], which uses a linear encoding method as the baseline. The results are reported when all algorithms are performed in a central node that considers all features given in the dataset. The DNN models are trained with Tensorflow [53], and we exploited Scikit-learn library [54] for the other algorithms. We exploit the common practice of grid search to identify the best hyper-parameters for each model. The accuracy of EdgeHD is reported for $D = 4000$ dimensions and 80% sparsity. Our evaluation shows that EdgeHD provides comparable classification accuracy to the sophisticated non-HD algorithms. As compared to the baseline HD computing, EdgeHD can achieve on average 4.7% higher classification accuracy since our new encoding method non-linearly maps the data to the high dimensional space, whereas the baseline HD performs the encoding in a linear way.

TABLE II
CLASSIFICATION ACCURACY IN HIERARCHY LEVELS.

	Centralized	Hierarchy-aware		
		End Nodes	Gateway	Central Node
PECAN	98.3%	59.5%	81.3%	98.3%
PAMAP2	97.3%	83.9%	91.2%	96.7%
APRI	92.6%	87.1%	89.3%	92.5%
PDP	94.6%	86.2%	89.5%	93.8%

Centralized vs. Hierarchical Learning: HD computing can be performed in a centralized or hierarchical way. In a centralized approach, all edge devices send all their data points to a cloud; then the cloud takes care of the encoding, training, and inference tasks. In contrast, in the proposed hierarchical EdgeHD, encoding, training, and inference are performed on all computing nodes in a distributed fashion.

Table II compares the accuracy of different applications for centralized and hierarchical learning cases. For hierarchical learning, we used the tree network topology that has three levels in the hierarchy. This classification accuracy increases as we go deeper through the hierarchy. For example, EdgeHD in the third level gets on average 94.4% accuracy, which has only a 0.4% difference from the average accuracy that the centralized model can get. Note that, in EdgeHD, the inference task can be performed in any node. Going deeper into the hierarchy increases the learning quality as the edge devices get more data to train their model. However, it comes at the expense of increasing computation and communication costs. As a result, the model located in the end node provides the lowest classification accuracy, i.e., on average, 85.7%, but it can be performed without any communication costs.

C. Hierarchical Online Learning

As discussed in IV-D, EdgeHD can update the models online based on negative feedback given by users. In each node, EdgeHD accumulates the hypervectors for the negative feedback in the residual hypervectors and propagates them periodically. We train the first offline model with 50% samples of the dataset and use the rest for online learning. We assume that each user gives negative feedback for the data points which are incorrectly classified.

PECAN: Figure 8 visualizes how EdgeHD processes PECAN consisting of 312 nodes connected through four hierarchical levels to a central node. Each node in the house level is connected to up to 12 appliances. On the next level, the street-level nodes collect information from 6-7 houses. Finally, the information on street-level nodes is collected in a central house-level node. We assumed the classification task can be performed on any nodes in the second to fourth levels (all levels except appliances). We propagate the models every midnight based on the timestamps given in the dataset.

As shown in Figure 8a, the initial model trained offline has relatively low classification accuracy, in particular, on the lower nodes. The central node has the highest accuracy since its model is trained by accessing all edge nodes' data. Online learning increases classification accuracy over all nodes. The increment is more significant on the lower-level nodes. Our evaluation shows that after 100% online learning, EdgeHD

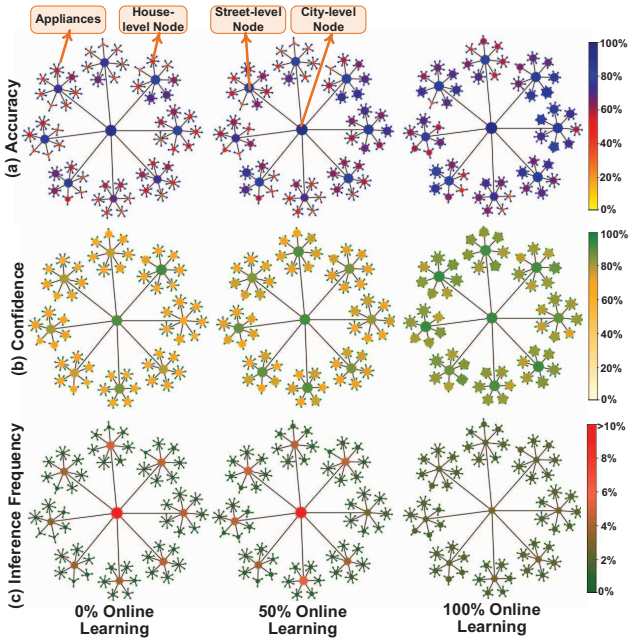


Fig. 8. Illustration of online learning (PECAN): (a) accuracy, (b) confidence level, and (c) inference frequency.

provides 59.5%, 81.3%, and 98.3% accuracy on the house-level, street-level, and city-level nodes, respectively. Similarly, the online training increases confidence level as shown in Figure 8b. Thus, as online learning progresses, EdgeHD can perform the inference more locally in lower-level nodes, saving communication costs. Figure 8c shows the frequency of each device selected to perform the inference tasks. Right after the offline training, many classifications are assigned to the central node since the lower-level devices have low confidence. For example, the central node performs 28.9% of the tasks. Through the online training, the classification happens in a highly distributed fashion, e.g., only 0.3% of prediction needs to happen on the central node after 100% online training.

Other datasets: Figure 9a shows the classification accuracy on the central node for PAMAP2. Our evaluation shows that, when using 4 steps, the online training with 50% and 100% of datasets improves the accuracy by 4.3% and 9.8% as compared to the initial model trained offline. The results also show that the more frequent propagation, the higher the final accuracy that EdgeHD can achieve. This comes with extra communication costs. However, it needs to send the residual hypervectors only once for each propagation, significantly reducing the communication costs. Figure 9b presents the classification accuracy over different steps for the four datasets when using ten steps. We observe that the online training successfully increases the accuracy on average by 5.5%.

D. Efficiency Comparison

We compare the computation efficiency of the DNN and HD computing algorithms. In this evaluation, we assume that the devices have access to the ideal network having 1Gbps bandwidth. We evaluate the STAR and TREE topology for each

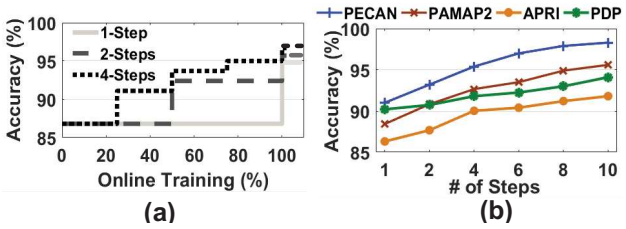


Fig. 9. EdgeHD accuracy in different training steps.

of the following configurations: (1) DNN-GPU performs the DNN learning on the GPU in the *centralized* way. The DNN models are trained with the best hyperparameter set obtained using the grid search. Note that there is no trivial efficient way to run the DNN algorithm in the hierarchy since each neuron of a DNN model communicates to all connected neurons during the backpropagation (training) and feed forwarding (inference.) (2) HD-GPU performs the proposed algorithm using the GPGPU implementation. Since the IoT devices may not typically have the general-purpose GPUs due to the costs, we assume that this approach can be deployed in the *centralized* way. (3) HD-FPGA performs the proposed algorithm in the *centralized* way using the proposed FPGA design. (4) EdgeHD is the proposed hierarchical learning method, where each node runs the learning procedure on the FPGA design and transfers the data using the host Raspberry Pi systems.

DNN vs HD in the centralized case: Figure 10 compares the efficiency of the training and inference procedure for the different configurations. All results are normalized to the execution time and energy consumption of DNN-GPU on the TREE topology. The results show that regardless of the algorithm, the centralized learning needs to pay the same amount of communication cost as the same number of data points need to be transferred to the central server. The communication cost depends on the network structure of the hierarchy, i.e., the TREE or STAR topology. In the tree-based configuration, the edge devices need to send their information to a server by sending their data through gateway devices. This data transfer requires extra time and energy for communication as compared to the STAR topology.

The results also show that DNN is a computationally expensive algorithm, thus requiring a significant amount of time and energy to train a model. In contrast, an HD-based classifier performs the training task at a much lower cost, even in the centralized case, while showing comparable classification accuracy. For example, HD-GPU achieves 4.3× (2.8×) and 10.5× (4.1×) reduction in the execution time and energy on average as compared to DNN-GPU in training (inference). HD-FPGA is slower than HD-GPU since the FPGA has lower computation resources as compared to the NVIDIA GPU. However, it achieves a higher energy saving of 3.0× than HD-GPU since it consumes lower power.

Hierarchical vs. Centralized: Although the HD-based learning method achieves higher efficiency than the DNN, even in the centralized case, this approach still has two main drawbacks. First, all the learning computation happens in a single node, potentially creating a scalability issue. Second, communication takes a large portion since all data needs to be sent through the network topology of the hierarchy. The proposed hierarchical learning approach addresses both computation and communication issues. For example, regarding the computation costs, EdgeHD training achieves 3.4× speedup and 3.9× higher energy efficiency as compared to HD-FPGA. It is because the lower-level nodes need to compute hypervectors of fewer dimensions, as discussed in Section IV-A. For example, the FPGA in the centralized setting consumes 9.8W on average, whereas the FPGA deployed in each node only needs 0.28W for the hierarchical case. The results show that EdgeHD also significantly saves the execution time spent for the communication, e.g., on average 85% for the training and 78% for the inference. EdgeHD reduces the communication costs by sending a small number of the class/batch hypervectors in training and compressing multiple hypervectors in the inference. In total, EdgeHD achieves 3.4× (1.9×) speedup and 11.7× (7.8×) energy efficiency improvement in the training (inference) as compared to HD-GPU. As compared to DNN-GPU, EdgeHD provides 14.7× (5.3×) speedup and 124.5× (43.6×) higher energy efficiency during training (inference).

E. Impact of Network Bandwidth

Figure 11 shows how the network bandwidths affect the efficiency of hierarchical learning. We report the speedup results when performing the inference tasks on the end node (Level-1), the gateway (Level-2), and the central node (Level-3), where HD-FPGA is used as the baseline. We have evaluated the EdgeHD performance efficiency on five network mediums: a wired network of 1Gbps, a wired network of 500Mbps, WiFi 802.11ac, WiFi 802.11n, and Bluetooth 4.0. The results show that when the network bandwidth is more limited, EdgeHD achieves higher speedup. For example, using 802.11ac with 46.5Mbps, EdgeHD can achieve on average 3.1× speedup as compared to centralized HD. This speedup increases to 9.2× when we use even lower bandwidth networks such as Bluetooth 4. In practical IoT systems, the network bandwidth can usually be limited. The recent Raspberry Pi 3 Model B+ uses WiFi 802.11ac and Bluetooth 4.0, which practically provide 23.5Mbps and 1MBps bandwidth, respectively. Therefore, EdgeHD is a suitable solution for IoT systems with limited bandwidth.

The evaluation also shows that performing the inference with the model at a lower level is faster than executing it on the last level node (Level-3). For example, the Level-2 inference operates 2.4× and 1.8× faster than the level-3 using the 802.11n and 1Gpbs bandwidth case, respectively. Thus, if the quality loss is acceptable, e.g., 3.2% accuracy difference between Level-2 and Level-3 for the APRI dataset, we may use the model at the lower level for higher efficiency.

F. Robustness to Failure

Figure 12 compares the classification accuracy, which is assumed to randomly lose a portion of bits in the encoded hypervector, and DNN, which loses feature values during

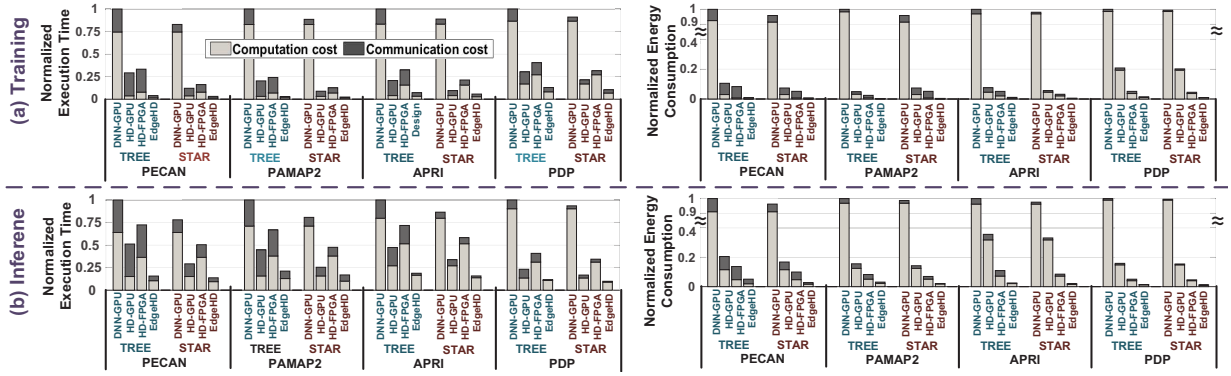


Fig. 10. Execution time and Energy of DNN and HD computing algorithm for centralized/hierarchical learning.

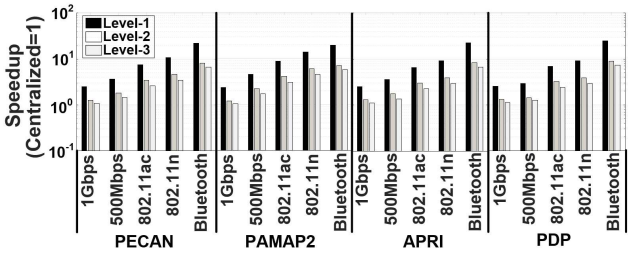


Fig. 11. Impact of network bandwidth on the performance speedup of hierarchical learning.

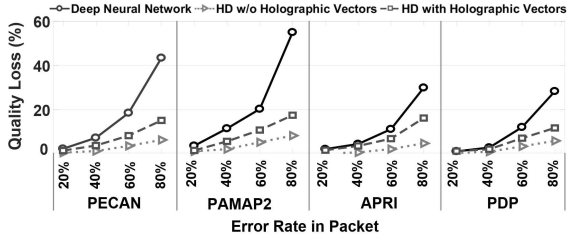
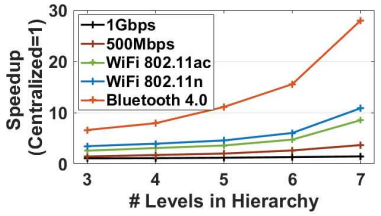


Fig. 12. Impact of the network and hardware failure on the classification accuracy of different applications.



Configuration		Accuracy
Centralized		98.3%
Hierarchal	3 Levels	98.3%
	4 Levels	98.3%
	5 Levels	98.2%
	6 Levels	98.1%
	7 Levels	97.9%

Fig. 13. Speedup and accuracy using configurations with different levels of hierarchy.

the data communication in the hierarchy. For EdgeHD, we have reported results for (i) the hierarchical encoding method explained in Section IV-A and (ii) a non-holographic encoding method that only concatenates the hypervectors without using the random projection method. The result shows that EdgeHD has much higher robustness to the possible data loss, especially with the holographic hierarchical encoding method. This robustness comes from the holographic mapping that distributes the feature information through all dimensions of the encoded data. For example, for the 80% loss case, the accuracy of the DNN model drops by up to 54.3%, while the maximum quality loss using the holographic (non-holographic) method is only 8.3% (17.5%).

G. Impact of Hierarchy Depths

We study the impact of the hierarchy depth on classification accuracy. Figure 13 shows the performance and classification accuracy of EdgeHD when the depth of the hierarchy increases from 3 to 7 levels for PECAN. The results show the performance speedup of EdgeHD compared to the centralized learning using the same configuration as hierarchical learning. Using configuration with more levels increases the cost of centralized and hierarchical learning, as both approaches need to transfer a more significant amount of data between the nodes. However, this data movement is much lower in EdgeHD since (i) the nodes only transfer limited data and (ii) the size of encoded data is reduced with more hierarchy levels. In addition, the increase in data transfer depends on the network bandwidth. As Figure 13a shows, EdgeHD using WiFi 802.11n (1Gbps) network can achieve 3.3× (1.2×) higher speedup than the centralized learning when the depth increases from 3 to 7 levels. In the hierarchical learning case, the greater improvement in the lower bandwidth stems from a larger amount of communication cost, which dominates the total computation cost. Figure 13b shows that EdgeHD can get similar accuracy over different levels. To provide the same dimensionality, EdgeHD with more levels needs to encode the data with lower dimensions at the end node, resulting in slightly lower accuracy. EdgeHD can compensate for this quality loss using a larger dimensionality in deep configurations.

VII. Conclusion

In this work, we propose EdgeHD, a hierarchy-aware learning algorithm that enables online learning through the hierarchy. EdgeHD enables both the training and inference to perform partially on embedded devices and significantly reduces the amount of data communication between the devices while accelerating the computation in a distributed manner. Our results also show that hierarchy-aware learning provides significantly higher efficiency when IoT systems have access to a low bandwidth network.

VIII. Acknowledgment

This work was supported in part by DARPA, National Science Foundation #2127780 and #2312517, Semiconductor

Research Corporation (SRC), Office of Naval Research, grants #N00014-21-1-2225 and #N00014-22-1-2067, the Air Force Office of Scientific Research under award #FA9550-22-1-0253, Army Ground Vehicle Systems Center, and generous gifts from Xilinx and Cisco. This work was also supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government(MSIT) (No.2018R1A5A1060031), Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Science. This work was supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government(MSIT) (No.2022-0-00991, 1T-1C DRAM Array Based High-Bandwidth, Ultra-High Efficiency Processing-in-Memory Accelerator).

REFERENCES

[1] G. Lopez *et al.*, "Edge-centric computing: Vision and challenges," *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 5, pp. 37–42, 2015.

[2] Y. Kang *et al.*, "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," *ACM SIGPLAN Notices*, vol. 52, no. 4, pp. 615–629, 2017.

[3] M. Satyanarayanan, "The emergence of edge computing," *Computer*, vol. 50, no. 1, pp. 30–39, 2017.

[4] B. P. Rimal *et al.*, "Mobile-edge computing vs. centralized cloud computing in fiber-wireless access networks," in *INFOCOM WKSHPS*, IEEE, 2016.

[5] N. Abbas *et al.*, "Mobile edge computing: A survey," *IEEE IoT Journal*, 2018.

[6] Y. Gan *et al.*, "An open-source benchmark suite for microservices and their hardware-software implications for cloud & edge systems," in *ASPLOS*, ACM, 2019.

[7] V. Smith *et al.*, "Federated multi-task learning," in *NIPS*, 2017.

[8] E. Bagdasaryan *et al.*, "How to backdoor federated learning," *arXiv preprint arXiv:1807.00459*, 2018.

[9] X. Wang *et al.*, "In-edge ai: Intelligentizing mobile edge computing, caching and communication by federated learning," *arXiv preprint arXiv:1809.07857*, 2018.

[10] J. Konečnỳ *et al.*, "Federated learning: Strategies for improving communication efficiency," *arXiv preprint arXiv:1610.05492*, 2016.

[11] G. Tzimpragos *et al.*, "Boosted race trees for low energy classification," in *ASPLOS*, ACM, 2019.

[12] G. Gobieski *et al.*, "Intelligence beyond the edge: Inference on intermittent embedded systems," in *ASPLOS*, pp. 199–213, ACM, 2019.

[13] "Huawei ai chip." https://www.engadget.com/2017/12/15/ai-processor-cpu-explainer-bionic-neural-npu/.

[14] N. L. HUYNH *et al.*, "Deepmon-building mobile gpu deep learning models for continuous vision applications," 2017.

[15] S. Yi, Z. Hao, Z. Qin, and Q. Li, "Fog computing: Platform and applications," in *2015 Third IEEE Workshop on Hot Topics in Web Systems and Technologies (HotWeb)*, pp. 73–78, IEEE, 2015.

[16] M. Aazam *et al.*, "Smart gateway based communication for cloud of things," in *ISSNIP*, IEEE, 2014.

[17] M. Aazam *et al.*, "Fog computing and smart gateway based communication for cloud of things," in *FiCloud*, IEEE, 2014.

[18] 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.

[19] Z. Zou *et al.*, "Scalable edge-based hyperdimensional learning system with brain-like neural adaptation," in *SC*, pp. 1–15, 2021.

[20] Y. Kim *et al.*, "Efficient human activity recognition using hyperdimensional computing," in *Proceedings of the 8th International Conference on the Internet of Things*, p. 38, ACM, 2018.

[21] A. Moin *et al.*, "A wearable biosensing system with in-sensor adaptive machine learning for hand gesture recognition," *Nature Electronics*, 2021.

[22] Z. Zou *et al.*, "Biohd: an efficient genome sequence search platform using hyperdimensional memorization," in *Proceedings of the 49th Annual International Symposium on Computer Architecture*, pp. 656–669, 2022.

[23] Y. Ni *et al.*, "Hdpg: Hyperdimensional policy-based reinforcement learning for continuous control," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, pp. 1141–1146, 2022.

[24] Y. Ni *et al.*, "Qhd: A brain-inspired hyperdimensional reinforcement learning algorithm," *arXiv preprint arXiv:2205.06978*, 2022.

[25] H. Chen *et al.*, "Darl: Distributed reconfigurable accelerator for hyperdimensional reinforcement learning," in *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*, pp. 1–9, 2022.

[26] P. o. Poduval, "Graphd: Graph-based hyperdimensional memorization for brain-like cognitive learning," *Frontiers in Neuroscience*, 2022.

[27] A. Rahimi, P. Kanerva, L. Benini, and J. M. Rabaey, "Efficient biosignal processing using hyperdimensional computing: Network templates for combined learning and classification of exg signals," *Proceedings of the IEEE*, vol. 107, no. 1, pp. 123–143, 2018.

[28] M. Imani *et al.*, "A framework for collaborative learning in secure high-dimensional space," in *IEEE CLOUD*, pp. 435–446, IEEE, 2019.

[29] Z. Zou *et al.*, "Eventhd: Robust and efficient hyperdimensional learning with neuromorphic sensor," *Frontiers in Neuroscience*, vol. 16, 2022.

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

[31] Z. Zou *et al.*, "Memory-inspired spiking hyperdimensional network for robust online learning," *Scientific Reports*, vol. 12, no. 1, p. 7641, 2022.

[32] A. Hernandez-Cane *et al.*, "Onlinehd: Robust, efficient, and single-pass online learning using hyperdimensional system," in *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 56–61, IEEE, 2021.

[33] N. P. Jouppi *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *ISCA*, pp. 1–12, IEEE, 2017.

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

[35] B. Scholkopf *et al.*, "Comparing support vector machines with gaussian kernels to radial basis function classifiers," *IEEE transactions on Signal Processing*, pp. 2758–2765, 1997.

[36] M. Imani *et al.*, "Hierarchical hyperdimensional computing for energy efficient classification," in *DAC*, p. 108, ACM, 2018.

[37] A. Rahimi *et al.*, "Hyperdimensional computing for blind and one-shot classification of eeg error-related potentials," *Mobile Networks and Applications* 2017.

[38] A. Rahimi *et al.*, "A robust and energy-efficient classifier using brain-inspired hyperdimensional computing," in *ISLPED*, ACM, 2016.

[39] E. P. Frady, D. Kleyko, C. J. Kymn, B. A. Olshausen, and F. T. Sommer, "Computing on functions using randomized vector representations," *arXiv preprint arXiv:2109.03429*, 2021.

[40] P. Kanerva *et al.*, "Random indexing of text samples for latent semantic analysis," in *Annual conference of the cognitive science society*, vol. 1036, Citeseer, 2000.

[41] Y. LeCun *et al.*, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, pp. 2278–2324, 1998.

[42] D. Ciregan *et al.*, "Multi-column deep neural networks for image classification," in *CVPR*, IEEE, 2012.

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

[44] D. Anguita *et al.*, "Human activity recognition on smartphones using a multiclass hardware-friendly support vector machine," in *International workshop on ambient assisted living*, 2012.

[45] Y. Vaizman, K. Ellis, and G. Lanckriet, "Recognizing detailed human context in the wild from smartphones and smartwatches," *IEEE Pervasive Computing*, vol. 16, no. 4, pp. 62–74, 2017.

[46] A. Angelova *et al.*, "Pruning training sets for learning of object categories," in *CVPR*, IEEE, 2005.

[47] "Pecan street dataport." https://dataport.cloud/.

[48] A. Reiss and D. Stricker, "Introducing a new benchmarked dataset for activity monitoring," in *ISWC*, pp. 108–109, IEEE, 2012.

[49] M. Zaharia *et al.*, "Apache spark: a unified engine for big data processing," *Communications of the ACM*, 2016.

[50] Y. Kim *et al.*, "P4: Phase-based power/performance prediction of heterogeneous systems via neural networks," in *ICCAD*, IEEE, 2017.

[51] T. R. Henderson, M. Lacage, G. F. Riley, C. Dowell, and J. Kopena, "Network simulations with the ns-3 simulator," *SIGCOMM demonstration*, vol. 14, no. 14, p. 527, 2008.

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

[53] M. Abadi *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *arXiv preprint arXiv:1603.04467*, 2016.

[54] F. Pedregosa *et al.*, "Scikit-learn: Machine learning in python," *JMLR*, 2011.