Scalable Edge-Based Hyperdimensional Learning System with Brain-Like Neural Adaptation

Zhuowen Zou[ψ], Yeseong Kim[†], Farhad Imani[‡], Haleh Alimohamadi[ψ]
Rosario Cammarota[††], and Mohsen Imani[§⋆]

[ψ]Uiversity of California San Diego, La Jolla, CA 92093, USA
[†]Daegu Gyeongbuk Institute of Science and Technology, South Korea
[‡]University of Connecticut, Storrs, CT 06269, USA
[††]Intel Labs, San Diego, CA 92131
[§]University of California Irvine, Irvine, CA 92697

[⋆]Corresponding Author: m.imani@uci.edu

ABSTRACT

In the Internet of Things (IoT) domain, many applications are running machine learning algorithms to assimilate the data collected in the swarm of devices. Sending all data to the powerful computing environment, e.g., cloud, poses significant efficiency and scalability issues. A promising way is to distribute the learning tasks onto the IoT hierarchy, often referred to *edge computing*; however, the existing sophisticated algorithms such as deep learning are often overcomplex to run on less-powerful and unreliable embedded IoT devices. Hyperdimensional Computing (HDC) is a brain-inspired learning approach for efficient and robust learning on today's embedded devices. Encoding, or transforming the input data into high-dimensional representation, is the key first step of HDC before performing a learning task. All existing HDC approaches use a static encoder; thus, they still require very high dimensionality, resulting in significant efficiency loss for the edge devices with limited resources. In this paper, we have developed NeuralHD, a new HDC approach with a dynamic encoder for adaptive learning. Inspired by human neural regeneration study in neuroscience, NeuralHD identifies insignificant dimensions and regenerates those dimensions to enhance the learning capability and robustness. We also present a scalable learning framework to distribute NeuralHD computation over edge devices in IoT systems. Our solution enables edge devices capable of real-time learning from both labeled and unlabeled data. Our evaluation on a wide range of practical classification tasks shows that NeuralHD provides 5.7× and 6.1× (12.3× and 14.1×) faster and more energy-efficient training compared to the HD-based algorithms (DNNs) running on the same platform. NeuralHD also provides 4.2× and 11.6× higher robustness to noise in the unreliable network and hardware of IoT environments as compared to DNNs.

SC '21, November 14–19, 2021, St. Louis, MO, USA
© 2021 Association for Computing Machinery.
ACM ISBN 978-1-4503-8442-1/21/11...$15.00
https://doi.org/10.1145/3458817.3480958

ACM Reference Format:
Zhuowen Zou, Yeseong Kim, Farhad Imani, Haleh Alimohamadi, Rosario Cammarota, and Mohsen Imani. 2021. Scalable Edge-Based Hyperdimensional Learning System with Brain-Like Neural Adaptation. In *The International Conference for High Performance Computing, Networking, Storage and Analysis (SC '21), November 14–19, 2021, St. Louis, MO, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3458817.3480958

1 INTRODUCTION

With the emergence of the Internet of Things (IoT), many applications run machine learning algorithms to perform learning and cognitive tasks. Today's systems typically rely on sending all the data to the cloud to complete learning and training, which leads to a significant communication cost [1]. This communication cost can be eliminated by scaling the learning tasks in a distributed fashion where different devices collect data. Edge computing tries to realize such a distributed computing paradigm by bringing the computation closer to the location where the data are generated [2]. A mainstream of the research is *federated learning* [2–5] that trains a central model over multiple devices. However, these techniques use complex algorithms, e.g., Deep Neural Network (DNNs), which require billions of parameters and many hours to train in powerful and reliable computing environment [6, 7]. Considering memory and resource limitations of embedded devices on edge, which also have potential issues of network noises and hardware failure due to the unstable nature of IoT systems, current computing environments are still far from real-time learning.

In contrast, the human brain can do much of this learning effortlessly and efficiently without much concern of noisy and broken neuron cells [8, 9]. To more closely model the human brain, earlier researchers proposed HyperDimensional Computing (HDC) as an alternative computing method, which mimics important brain functionalities towards high-efficiency and noise-tolerant computation [9–11]. HDC is motivated by the observation that the human brain operates on high-dimensional representations of data. In HDC, objects and data are thereby encoded with high-dimensional vectors, called *hypervectors*, which have 10,000 or more elements [10]. HDC can then perform various learning tasks with computation in the high dimensional space. HDC is well suited to address learning on edge systems as: (i) HDC models are computationally efficient to train, highly parallel at heart, and amenable to hardware level optimization [12, 13], (ii) HDC model offers an intuitive and human-interpretable model [14–16], (iii) it offers a complete computational paradigm that can be applied to cognitive as well as learning problems [8, 14, 17–22], and (iv) it provides strong robustness to noise [12, 23, 24] – a key strength for IoT systems, and (v) HDC can naturally enable secure learning [25, 26]. These features make HDC a promising solution for today's embedded devices with limited storage, battery, and resources, as well as future computing systems in deep nano-scaled technology, which devices will have high noise and variability.

Encoding, or transforming the input data into high dimensional representation, is the key first step to map data into the high dimensional space [22, 25, 27–29]. Mathematically, HDC requires huge dimensionality to provide acceptable accuracy on complex learning tasks. However, increasing dimensionality of the state-of-the-art HDC solutions results in efficiency loss and scalability issues: (i) In high-dimension, HDC involves a large amount of computation, resulting in a significant drop in learning efficiency. (ii) In a distributed system, dimensionality increases data size and communication costs, which are the dominant portion of system efficiency [25]. We observe that the root cause of why HDC requires high dimensionality is that the existing encoding modules often fail to find a good representation of the data. During learning, all dimensions of encoded data do not have the same impact on learning. Many dimensions are insignificant and can be dropped or replaced with no impact on the learning quality. Unfortunately, the existing HDC algorithms are using a static encoder with no capability of detecting the importance of dimensions [14, 17, 25, 27]. One of the primary reasons that HDC algorithms still require massive dimensionality is to capture all possible relations between the input features.

In this paper, we proposed NeuralHD, a novel *dynamic* and *adaptive* Hyperdimensional learning system. Our design is inspired by several research works in neuroscience [30–34], which present the neurons in the human brains are dynamically changing as shown by every day that thousands of neurons (brain cells) die and replace by new ones to give a more useful functionality to the brain [35, 36]. During learning, NeuralHD identifies insignificant dimensions and regenerates those dimensions to enhance learning capability, hence reducing the required dimensionality and improving its learning efficiency significantly. In contrast to the existing methods that use the static encoder to map each input feature into the hyperspace, the proposed solution dynamically regenerates dimensions, delivering high efficiency and robust learning against hardware failure and noise, as its primary advantages for IoT-edge platforms. The main contributions of the paper are listed below:

- To the best of our knowledge, **NeuralHD is the first HDC algorithm with dynamic and regenerative encoding module.** NeuralHD identifies insignificant dimensions in an unsupervised way and regenerates them to enable dimensions to have a more positive impact on the learning process. NeuralHD ensures the learning task to perform in a very high effective dimensionality while keeping the physical dimensionality significantly smaller.
- **NeuralHD introduces two learning approaches to support dimension regeneration**: (i) *Reset learning* that trains a new model after every regeneration phase, (ii) *Continuous learning* that exploits prior knowledge during the model training. Although *reset learning* is designed for high accuracy, *continuous learning*, inspired by human neural adaptation [30, 32], provides fast and affordable learning, which is desirable for embedded devices on edge.
- **We present a scalable learning framework to distribute NeuralHD computation over edge devices.** We propose a single-pass training approach that makes edge devices capable of real-time learning from labeled and unlabeled data. Our solution significantly reduces the amount of data communication between edge clouds, resulting in high system efficiency.
- **We show the high robustness of NeuralHD to noise in hardware and networked edge environment.** Since NeuralHD encoding spreads the data over a very large hypervector, a substantial number of bits can be corrupted while preserving sufficient information, resulting in high noise robustness.

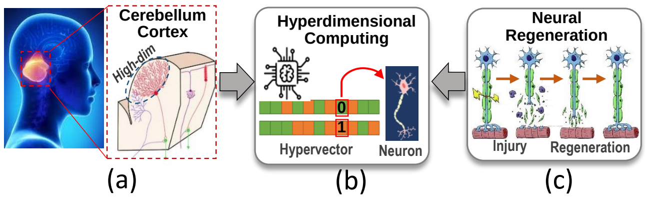


Figure 1: Hyperdimensional computing: model of brain cerebellum cortex

We evaluate NeuralHD on a wide range of practical classification tasks. Our evaluation shows that NeuralHD achieves much better accuracy than the existing HDC algorithm and comparable accuracy to sophisticated learning algorithms such as SVM and DNN. In terms of efficiency, NeuralHD significantly reduces the hypervector dimensionality, resulting in high learning efficiency. For example, NeuralHD provides, on average, 5.7× and 6.1× (12.3× and 14.1×) faster and more energy-efficient training as compared to the state-of-the-art HDC algorithms (DNNs). NeuralHD also provides 4.2× and 11.6× higher robustness to noise in the network and hardware than DNNs [1].

2 HYPERDIMENSIONAL COMPUTING

The cerebellum cortex is an area in the brain that plays a significant role in cognitive functions (shown in Figure 1a). This area of the brain stores information as a pattern of neural activity in a Purkinje cell layer (Figure 1a). HyperDimensional Computing (HDC) emerged from theoretical neuroscience as a short-term human memory model [9, 10, 37, 38]. HDC is motivated by the understanding that the cerebellum cortex operates on high-dimensional representations of data that originated from the large size of brain circuits (Figure 1b). It thereby models the human memory using points of a high-dimensional space, that is, with hypervectors. As Figure 1b shows, each dimension of hypervector models a neuron functionality at an abstract level. In high-dimensional space, there exist a huge number of nearly orthogonal hypervectors [10, 39]. This enables us to combine such hypervectors using well-defined operations while keeping the information of the two with high probability.

2.1 Encoding

Figure 2 shows an overview of the HDC classification task. Regardless of the learning task, the first step of HDC is to encode input

[1]https://gitlab.com/biaslab/neuralhd

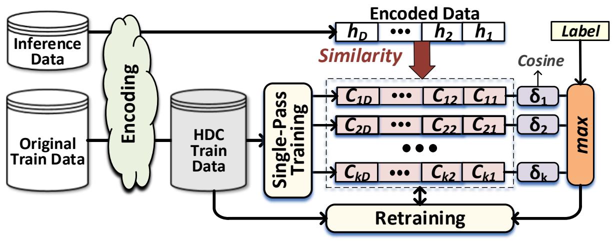


Figure 2: Overview of HDC Classification.

data into high-dimensional space, called *hypervectors*. A hypervector stores all the information across all its components so that no component is more responsible for storing any piece of information than another. HDC uses different encoding methods depending on the data type. Let us assume $\{\vec{L}_A, \vec{L}_B, \vec{L}_C, \vec{L}_D\}$ as randomly generated hypervectors ($\vec{L}_i \in \{-1,+1\}^D$). These hypervectors are nearly orthogonal, meaning that: $\delta(\vec{L}_A, \vec{L}_B) \approx 0$, where δ denotes the cosine similarity. The HDC encoding is based on the following set of primitives:

Bundling (+): is defined as an element-wise addition of multiple hypervectors: $H = \vec{L}_A + \vec{L}_B + \vec{L}_C$. The result of bundling is another hypervector with the same dimensionality as inputs. In high-dimensional space, bundling is like a memory operation where the bundled hypervector remembers the input operands' information. For example, we can check the existence of $\vec{L}_A$ and $\vec{L}_R$ on bundled $\vec{H}$ using: $\delta(\vec{H}, \vec{L}_A) >> 0$ and $\delta(\vec{H}, \vec{L}_D) \approx 0$.

Binding ($*$) : associates the information of multiple objects into a single hypervector. A binding defines as a bitwise XOR operation in binary, and multiplication operation in the bipolar domain ($\vec{H} = \vec{L}_A * \vec{L}_B$). The binded hypervector is a new object in high-dimensional space which is orthogonal to all input hypervectors ($\delta(\vec{H}, \vec{L}_A) \simeq 0$ and $\delta(\vec{H}, \vec{L}_B) \simeq 0$).

permutation (ρ): is defined as a rotational shift operation. A single permutation of a random hypervector generates a new hypervector with nearly orthogonal representation: $\delta(\vec{L}_A, \rho\vec{L}_A) \approx 0$. The permutation is suitable for preserving a sequence.

2.2 HDC Learning

Training: after generating each encoding hypervector $\vec{H}^l$ of inputs belonging to class/label l, the class hypervector $\vec{C}^l$ can be obtained by bundling (adding) all $\vec{H}^l$s. Assuming there are J inputs having label l: $\vec{C}_l = \sum_j^J \vec{H}_j^l$

Retraining can boost the accuracy of the HDC model by discarding the mispredicted queries from corresponding mispredicted classes, and adding them to the right class. Retraining examines if the model correctly returns the label l for an encoded query $\vec{H}$. If the model mispredicts it as label l', the model updates $\vec{C}_l$ and $\vec{C}_{l'}$ as follows:

$$\vec{C}_l = \vec{C}_l + \vec{H} \qquad \& \qquad \vec{C}_{l'} = \vec{C}_{l'} - \vec{H} \tag{1}$$

Inference of HDC has a two-step procedure. The first step encodes the input (similar to encoding during training) to produce a query hypervector $\vec{H}$. Thereafter, the similarity (δ) of $\vec{H}$ and all class hypervectors are obtained to find out the class with the highest similarity. Depending on the precision of the hypervectors, NeuralHD can use different metrics for similarity search. In binary representation, Hamming distance is a proper similarity metric, while for hypervectors with high precision, the cosine distance is used as a similarity metric.

2.3 Challenges

Several research works in neuroscience have shown that neurons in the brains are dynamically changing. Every day 85,000 neurons (brain cells) die; that's 31 million in a year [40, 41]. Simultaneously, a similar number of neurons are generating to give more useful functionality to the brain. The generated neurons try to learn new information and help the entire brain system for innovation and adaptive learning. Figure 1c shows brain neural regeneration Hyperdimensional computing, as a brain-inspired computing approach, needs to support a similar behavior. The goal of HDC is to exploit the high-dimensionality of randomly generated base hypervector to represent the information as a pattern of neural activity. Each dimension of the encoded hypervector, in the abstract level, represents a neuron in the brain. Unfortunately, all existing HDC algorithms are using a static encoder. These approaches generate base hypervectors with fixed values and use those bases for the rest of the learning task. All dimensions equally contribute to the learning task. However, in practice, data points and environments are dynamically changing. Thus, they require many dimensions to capture the relation between the input features. However, a large dimensionality has a direct impact on HDC inefficiency. Our goal is to design a dynamic HDC encoder that identifies insignificant dimensions and replace them with new dimensions with a more positive impact on the accuracy.

3 PROPOSED NEURALHD

3.1 NeuralHD Overview

In this paper, we proposed NeuralHD, a novel HDC approach with a dynamic encoder for adaptive learning. In HDC, all dimensions do not have a similar impact on the learning task; There are some dimensions/neurons with no or minimal impact on learning. The goal of our proposed NeuralHD is to identify such insignificant dimensions and drop them from the computation. To enhance accuracy, NeuralHD extends the model by regenerating those dimensions on the encoding module. This regeneration gives a new chance to dimensions to have a higher contribution to the learning task.

Figure 3 shows an overview of NeuralHD learning framework. NeuralHD first maps data into high-dimensional space using one of the existing encoding methods (A). The encoding depends on the data type and is performed using defined HDC mathematics over a set of randomly generated base vectors (e.g., for text classification as the definition of A to Z alphabets as random hypervectors in $D = 10k$ dimensions). We explain the details of this static encoder later in Section 3.3. NeuralHD performs training over the encoded data (B). The trained model is normalized to simplify the similarity metric used in the inference/retraining to dot product operation (C) (explained in Section 3.2). Next, we compute the variance over different class dimensions in order to find dimensions with the lowest impact on classification accuracy (D). NeuralHD drops insignificant dimensions from our model and base hypervectors in the encoder module (E)). Finally, NeuralHD regenerates the base hypervectors on the selected dimensions (F). NeuralHD encodes the data points into high-dimensional space using new update base vectors. To continue the learning, NeuralHD introduced

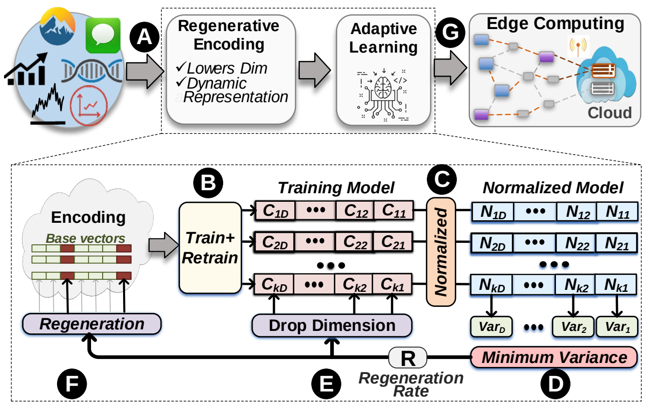


Figure 3: overview of NeuralHD using regenerative encoder and adaptive learning.

two iterative learning to repeat training, dimension reduction, and dimension regenerating (explained in Section 3.4). The iterative learning and regeneration continue until NeuralHD finds a model that most dimensions are highly contributing to the classification (explained in Section 3.6). We also exploit NeuralHD compressed and efficient model to enable online learning on edge computing systems (G).

3.2 Drop Insignificant Dimensions

During training, HDC creates a single hypervector representing each class. The inference task happens by checking the similarity of a query hypervector (an encoded inference data) with all class hypervectors. Query data is assigned to a class with the highest similarity. The goal of HDC is to train class hypervectors, where their patterns represent information. A weak classifier cannot find distinct patterns for different classes; thus, many classes will get similar patterns. This makes the classification task hard as the query may have a close similarity value to multiple classes.

Figure 2 shows how HDC computes the similarity of a query, $\vec{H} = \{h_1, h_2, \cdots, h_D\}$, with multiple class hypervectors. During learning, HDC computes the cosine similarity as:

$$\delta(\vec{H}, \vec{C_l}) = \frac{\vec{H} \cdot \vec{C_l}}{||\vec{H}|| \cdot ||\vec{C_l}||} = \frac{\vec{H}}{||\vec{H}||} \cdot \frac{\vec{C_l}}{||\vec{C_l}||} \cong \vec{H} \cdot \vec{N_l} \qquad (2)$$

where $||\vec{H}||$ is a repeating factor when comparing a query with all classes, so can be discarded. The $||\vec{C^l}||$ factor is also constant for a classes, so only needs to be calculated once. This simplified the cosine similarity to dot product operation.

The same product operation happens between a query and all other classes. As Figure 2 shows, in each dimension, the same query data is multiplied to all class elements on that dimension. Our goal is to find dimensions that do not have a significant impact on the classification task. To this end, as Figure 3D shows, we compute the variance over each dimension of the classes. Dimensions with low variance have similar values over all classes, meaning that they store *common information*. During the search, these dimensions are adding similar weights to cosine over all classes. We call these dimensions insignificant as they do not help to differentiate the class patterns.

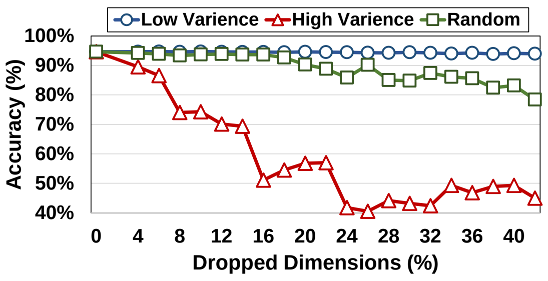


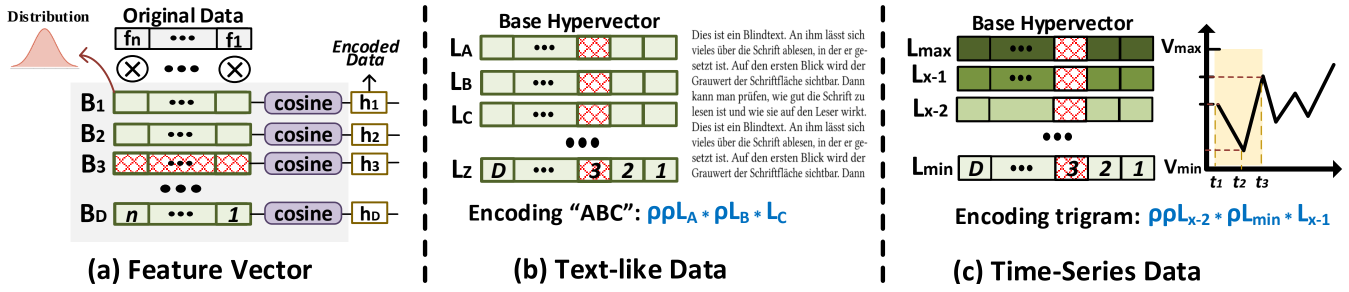
Figure 4: Dropping dimensions and NeuralHD accuracy.

NeuralHD drops dimensions with low variance (Figure 3E). Figure 4 shows the impact of dimension reduction on NeuralHD classification accuracy. In our evaluation, we drop hypervector dimensions in three configurations: (i) dimensions with the lowest variance, (ii) random dimensions, and (iii) dimensions with the highest variance. Our evaluation shows that dropping low variance dimensions has almost no impact on the accuracy while dropping higher variance dimensions results in a significant accuracy drop. Similarly, drooping random dimensions are providing a medium impact on the accuracy drop. This indicates that the low variance dimensions are insignificant and have minimal impact on accuracy.

3.3 NeuralHD Dimension Regeneration

NeuralHD drops insignificant dimensions and regenerates them during the training phase. Figure 3F shows how this regeneration happens over a dropped dimension. During the training phase, NeuralHD creates an initial HDC model, a hypervector representing each class. For each dimension of class hypervector, NeuralHD computes the variance over the normalized model (shown in Figure 3D). Then, NeuralHD selects R% of dimensions with minimum variance as a candidate to drop, where R is a regeneration rate. Instead of leaving the drooped dimensions blank, NeuralHD regenerates those dimensions. The main goal of regeneration is to create new dimensions that can potentially have a higher impact on the classification, meaning that they can provide a higher variance. Note that NeuralHD regeneration is a general approach and applicable to all existing HDC algorithms. In this paper, we look at its impact on the classification task. In the following, we explain the encoding methodology for popular data types and NeuralHD modifications during the regeneration phase.

Feature Data: We exploit an encoder method, inspired by the Radial Basis Function (RBF) kernel trick [42, 43], for mapping data points into HDC space. This encoder considers the non-linear relation between the features during the encoding. Figure 5a shows our encoding procedure. Let us consider an encoding function that maps a feature vector $\vec{F} = \{f_1, f_2, \ldots, f_n\}$, with n features ($f_i \in \mathbb{R}$) to a hypervector $\vec{H} = \{h_1, h_2, \ldots, h_D\}$ with D dimensions ($h_i \in \{0, 1\}$). We generate each dimension of encoded data by calculating a dot product of 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 a randomly generated vector from 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 $\{\vec{B}_1, \vec{B}_2, \cdots, \vec{B}_D\}$ can be generated once offline and then can be used for the rest of the classification task ($\vec{B_i} \in \mathbb{R}^m$). After this step, each element, h_i, represents an element of $\vec{H}$ hypervector.



Dies ist ein Blindtext. An ihm lässt sich vieles über die Schrift ablesen, in der er gesetzt ist. Auf den ersten Blick wird der Grauwert der Schriftfläche sichtbar. Dann kann man prüfen, wie gut die Schrift zu lesen ist und wie sie auf den Leser wirkt. Dies ist ein Blindtext. An ihm lässt sich vieles über die Schrift ablesen, in der er gesetzt ist. Auf den ersten Blick wird der Grauwert der Schriftfläche sichtbar. Dann

Figure 5: NeuralHD encoding and dimension regeneration over different data types.

Regeneration: As Figure 5a shows, each base vector in the encoding module corresponds to generating a single dimension of hypervector. For instance, the i^{th} dimension of encoded data is generated by performing dot product operation between the feature vector and B_i. NeuralHD regenerates a selected dimension on the encoding module by updating the corresponding base vector with another randomly generated vector with a Gaussian distribution. The same base update happens on all dimensions that are selected to drop from the class hypervectors.

Text-like Data: The encoding of the text data starts by assigning a random binary hypervector to each character in the alphabet. For example, for encoding the English text, we generate a random hypervector representing digits A to Z (as shown in Figure 5b). We encode text data in a n-gram windows, where n is usually a number between 3 to 5 [27]. Considering a trigram "A-B-C" example, we use the following embedding to map a sequence to high-dimensional space: $\rho\rho\vec{L}_A * \rho\vec{L}_B * \vec{L}_C$. The encoding module binds the hypervectors corresponding to alphabets while exploits permutation to remember their sequence.

Regeneration: Due to the rotational shift that happens by the permutation, the location of dimensions in the base hypervectors do not linearly match with the class dimensions on the trained model. Assuming an n-gram windows, a change on the i^{th} dimension of base hypervectors can result in changes on i^{th} to $i+n^{th}$ dimensions of the class hypervectors. Instead of looking for a single dimension of the model with low variance, NeuralHD finds the n neighbor dimensions that have the lowest average variance. NeuralHD regeneration updates the i^{th} dimension on the base hypervectors, if class dimensions in i^{th} to $(i+n)^{th}$ have minimum average variance. This update happens by generating random uniform bits on the i^{th} dimension of all base hypervectors.

Time-Series Data: HDC uses a very similar encoding as text data to map time-series into high-dimensional space. We sample time-series in an n-gram window. In each sample window, the signal values (in the y-axis) stores the information, and the time (x-axis) represents the sequence. We assign a random vector to V_{min} ($\vec{L}_{min}$ representing minimum signal value) and V_{max} ($\vec{L}_{max}$ representing maximum signal value). Since these vectors are randomly generated, they are nearly orthogonal. For signal values between V_{min} and V_{max}, we perform vector quantization to generate vectors that have a spectrum of similarity to $\vec{L}_{min}$ and $\vec{L}_{max}$ similarity. Finally, the encoding can perform by binding the level hypervectors corresponding to sampled signal while using permutation to store the timing information. For example shown in Figure 5c, the trigram Windows can be encoded as $\rho\rho\vec{L}_{x-2} * \rho\vec{L}_{min} * \vec{L}_{x-1}$.

Regeneration: Similar to text-data, NeuralHD selects insignificant dimensions by computing the average variance over n neighbor dimensions of class hypervectors . Assuming i^{th} to $i+n^{th}$ class dimensions have the minimum average variance, NeuralHD drops and regenerates the i^{th} dimension on the $\vec{L}_{min}$ and $\vec{L}_{max}$. The intermediate levels ($\vec{L}_{x-1}$, and $\vec{L}_{x-2}$ shown in Figure 5C) are computed by performing vector quantization between $\vec{L}_{min}$ and $\vec{L}_{max}$.

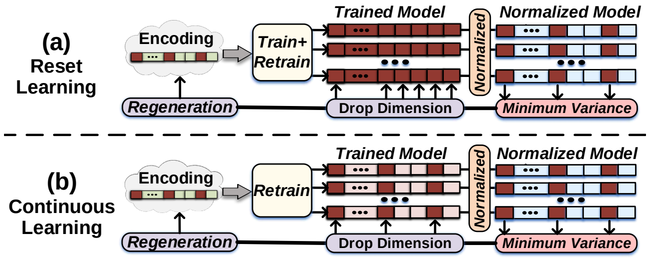


Figure 6: NeuralHD reset and continuous learning.

3.4 NeuralHD Retraining

The regeneration updates the encoding module. However, NeuralHD current model has been trained based on the old bases in the encoding module. To adapt model to count for regeneration, NeuralHD proposes two re-training approaches: *Reset Learning* and *Continues Learning*. Reset learning starts training a new model based on the regenerated bases, while *Continues learning* exploits the prior knowledge to continue the learning procedure.

3.4.1 Reset Learning. Figure 6a shows an overview of reset learning. NeuralHD starts training a model from scratch using new regenerated bases. This approach does not exploit the information learned by the model in the previous iteration. The training happens by exploiting the new encoded data points and training a hypervector representing each class. Thanks to newly updated base hypervectors, NeuralHD is expected to get a larger variance on the regenerated dimensions. This indicates that more dimensions are becoming significant and will have a higher impact on the classification. Due to the randomness of the regeneration, it is possible that the newly updated dimension still provides low variance. NeuralHD iterative training gives the same change to updated as well as all other dimensions to drop again during the next regeneration iterations. This ensures that NeuralHD can find suitable dimensions during the iterative process. Although this approach improves classification accuracy, it is a slow learning process. Since NeuralHD starts each learning iteration from a scratch model (losses the past information), it requires several iterations to converge.

3.4.2 Continuous Learning. Figure 6a shows an overview of continuous learning. NeuralHD continues learning from the previously learned model. Instead of start training from scratch, NeuralHD only ignores the class values on the dropped dimensions while other dimensions continue learning at the top of their existing values. In each iteration, NeuralHD checks the similarity of each training data point with the current model. If the training data point correctly classified with the current model, it does not update the model. However, in the case of miss-classification, NeuralHD updates the model by adding the query with correct and subtracting it from the wrong class. This process continues over the entire dataset (or a batch of data) to generate a new model. At the end of the iteration, the new model can be used for the inference task or another iteration of retraining.

3.5 Similarity To Brain Regeneration

NeuralHD continuous training has a similar behavior as human neural adaptation, where insignificant neurons die, and newborn neurons perform the same functionality. Like the human brain that evolves more quickly at the young ages (learning phase), NeuralHD has more rapid regeneration during the first regeneration iterations. Figure 7a visually shows the index of the regenerated dimensions during different iterations (white dots show the regenerated dimensions). In the initial regeneration iterations, our approach effectively identifies various dimensions for regeneration. Going further through the regeneration, NeuralHD effectively drops insignificant dimensions such that the remaining dimensions are getting a higher variance. Figure 7b shows the average variance of the class hypervector over different dimensions. The results indicate that NeuralHD regeneration drops insignificant dimensions and helps to find dimensions that can significantly impact classification. The increase in the variance depends on the regeneration rate, where higher rates result in a higher variance.

As Figure 7a shows, the new regenerated dimensions in the last iterations (> 30) will find less chance to compete with already well-developed dimensions in terms of variance. Therefore, NeuralHD may repeatedly select the recently regenerated dimensions as the best candidate to drop. Biologically, this is a very similar behavior that neuroscientists observed as the functionality of the human brain [44]. During childhood, the brain regenerates more neurons, while this process slows down by getting aged when the brain gets more complex [41, 45].

3.6 NeuralHD Learning Convergence

As we explained, NeuralHD is an iterative learning approach that regenerates dimensions through iterations. However, the convergence of NeuralHD highly depends on the frequency of the model update and the regeneration rate. Frequently updating the encoding bases can result in model divergence.

Lazy Regeneration: The newly updated dimensions still have very low variance, as these dimensions only retrained for a single iteration. In contrast, other dimensions have been retrained for

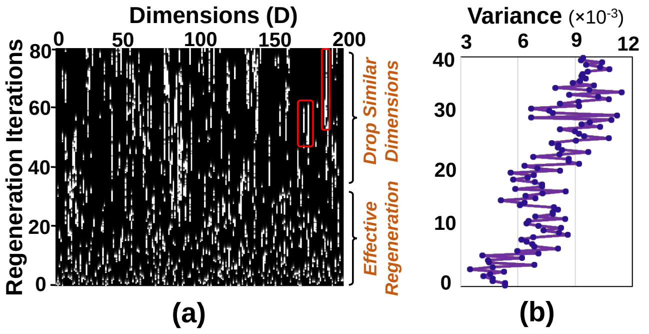


Figure 7: (a) Visualizing the index of regenerated dimensions during different iterations, (b) Average dimensions variance.

multiple iterations. During the next regeneration phases, the recently updated dimensions have a higher chance of being selected to drop from the model again. To address this issue, we perform less frequent regeneration, called *lazy regeneration*, rather than updating the encoder in every iteration. This technique ensures that: (i) NeuralHD accuracy does not diverge due to aggressive model update, and (ii) regenerated dimensions have enough chance to increase their variance during the iterative learning process.

Weighting Dimensions: Another important issue with model regeneration is the small class values on newly generated dimensions. NeuralHD training and retraining process increases the values of the class elements. However, the newly updated dimensions usually have small values. During the cosine similarly, the impact of each dimension is determined by its value in each dimension. This means that newly updated dimensions do not have a major impact on the cosine values. To give a similar chance to new dimensions to contribute to the similarity distance, we normalize the class hypervectors over each dimension (Figure 3C). This normalization puts all dimension values in a similar range as newly generate dimensions. Therefore, all dimensions will have the same chance during the similarity measurement. Note that this normalization happens after every m iterations when regeneration happens. During the inference, NeuralHD performs the exact similarity search as the baseline to find a class with the highest similarity.

4 NEURALHD EDGE LEARNING

In IoT systems, devices are connected as a network. A centralized could often receive data from multiple end-node devices, called edge devices. Several research works focused on using neural networks for efficient distributed learning, e.g., federated learning [2–4, 46]. However, these approaches have the following challenges: (1) Edge devices are often lightweight embedded devices with limited battery, memory, and computing resources. Thus, they cannot provide real-time DNN training, which requires a large memory footprint and computing resources to perform gradient computation. (ii) They suffer from a lack of robustness in both hardware and network noise.

In contrast, NeuralHD provides several features that make it suitable for learning in edge learning: (i) fast and real-time learning from the stream of data, and (ii) robustness to noise and failure on the end-node devices and the network. We exploit NeuralHD for both centralized and federated learning. In centralized learning, each edge device encodes data into high-dimensional space.

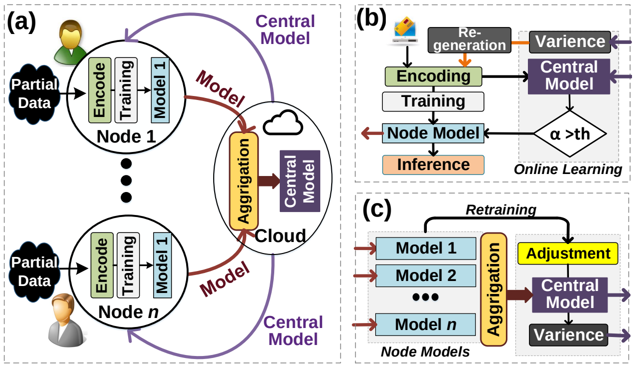


Figure 8: NeuralHD federated learning. (a) Overview, (b) Node computation, (c) Cloud computation.

All devices send their encoded hypervector to a cloud, and cloud performs learning tasks (learning explained in Section 2.2). The trained model will be shared with all edge devices. Although this approach provides high classification accuracy, it requires a large amount of communication cost, which dominates the total system efficiency. In Section 6.2 and Section 6.4, we evaluate NeuralHD accuracy and efficiency in this configuration.

4.1 NeuralHD Federated Learning

Figure 8a shows an overview of NeuralHD framework supporting federated learning. In federated learning, the encoding and training tasks perform locality on edge devices. Cloud creates an aggregated model by combining models received by all nodes. Then, it shares the aggregated model with all nodes. Each node made necessary changes to personalize the model based on its local data. The updated models of all edge devices will be aggregated on the cloud again. This iterative learning continues until generating a central model with the best representation of data received by all nodes. In the following, we explain the details of this iterative process.

Edge Learning: Figure 8b shows the functionality of each edge device during federated learning. Each edge device encodes data points into high-dimensional space and trains a local model. NeuralHD supports both iterative learning (explained in Section 2.2) or fast single-pass training (explained in Section 4.2). In the first iteration, the learning happens without regenerating the encoding module. NeuralHD also supports online learning from both labeled or unlabeled data. More details are explained in Section 4.2.

Cloud Aggregation: Figure 8c shows the functionality of the cloud during federated learning. Cloud receives partially trained models from all edge devices and aggregates them to create a central model. Assume a system with m edges, where $\{\vec{C}_i^1, \vec{C}_i^2, \cdots, \vec{C}_i^m\}$ are the i^{th} class hypervectors corresponding to different nodes. NeuralHD aggregates models by adding different patterns corresponding to each node. For example, the i^{th} class of the central model can be created using: $\vec{C}_i^A = \vec{C}_i^1 + \vec{C}_i^2 + \cdots + \vec{C}_i^m$. However, the central model is often saturated by the pattern of dominant nodes, and it does not well represent the entire data. NeuralHD addresses this issue by retraining the aggregated model over the class hypervectors received by all nodes. We consider each class hypervector as labeled encoded data. We check the similarity of each class hypervector with the aggregated model. For example, we check the similarity of the i^{th} class hypervector of the node 1 with the aggregated model ($\delta(\vec{C}_i^A, \vec{C}_i^1)$, $i \in 1...k$). For every incorrect prediction, NeuralHD updates the corresponding class hypervector in the central model as follows: $\vec{C}_i^A \leftarrow \vec{C}_i^A + (1 - \delta(\vec{C}_i^A, \vec{C}_i^1)) \times \vec{C}_i^1$, where '$1 - \delta(\vec{C}_i^A, \vec{C}_i^1)$' term ensures that models do not saturate the class if their pattern already exists in the aggregated model. Cloud continues this retraining in multiple iterations until generating a good representative model.

Cloud Dimension Selection: Cloud performs dimension reduction over the aggregated model, while the regeneration happens locally on each node. Cloud computes the variance on different dimensions of the aggregated model. Then, it sends the central model along with the index of dropped dimensions (variance vector) to all nodes (Figure 8b,c).

Edge Personalized Training: As Figure 8b shows, NeuralHD regenerates the base vectors on insignificant dimensions chosen by the cloud. All nodes update the central model locally based on their data points. This is equivalent to personalizing the model in each node. NeuralHD supports model adjustment in both iterative or streaming ways. In the iterative procedure, each node iterates over the same training data point and updates the model (Section 2.2). NeuralHD is also capable of single-pass training to update the model based on the stream of data received by each node. We explain the details of NeuralHD single-pass training in Section 4.2.

Inference and Continuous Learning: The new model in each node can be used for the inference. To further improve the classification accuracy, NeuralHD can continue collaborative learning by aggregating all nodes' models in the cloud. The rest of the learning task repeats as the steps explained above. While NeuralHD tries to generate a new aggregated model, edge devices can perform inference based on their most updated model.

4.2 Online Learning on the Edge

One of the main features of HDC is its capability to support single-pass training. NeuralHD exploits this feature to enable online learning. NeuralHD learns a model by one-time passing through each data point, with no iterations. This enables efficient learning without storing train data. The single-pass training is suitable for many real-world IoT systems where the learning is happening on small embedded devices with limited memory and resources.

Besides supervised learning, NeuralHD also supports online learning from semi-supervised data, where only a small portion of training data is labeled. NeuralHD first creates a model using labeled data. Then, it exploits unlabeled data to improves the quality of the model. NeuralHD checks the similarity of each unlabeled data with the already trained model (Figure 8b). Thanks to NeuralHD transparent model, the similarity search gives us confidence about the classification result. The confident level (α) determines how much the model ensures that a data is corresponded to a particulate class. Assume δ_i is a cosine similarity of an encoded data with i^{th} class ($\delta_i = \delta(\vec{H}, \vec{C}_i)$), we compute the confidence of class i^{th} as: $\alpha_i = \frac{\delta_{max \neq i} - \delta_i}{\delta_{max \neq i}}$, where $\delta_{max \neq i}$ is the class with the maximum similarity to query except with i^{th} class. If the confident level is higher than a threshold (e.g., $\alpha > 90\%$), NeuralHD updates the model by embedding encoded data to the corresponding class hypervector as: $\vec{C}^{max} = \vec{C}^{max} + \alpha \times \vec{H}$, where $\vec{H}$ is the query data and $\vec{C}^{max}$

is a class which has the maximum similarity with a query. This approach enables model updates using unlabeled data.

NeuralHD supports regeneration during single-pass training. The dimension reduction used for regeneration is unsupervised, where we can identify insignificant dimensions regardless of the labeled data. During training, NeuralHD computes the variance and drops dimensions for the next set of training data points. It then regenerates the base vectors in the encoding to replace insignificant dimensions. NeuralHD uses a very low regeneration rate to ensure model convergence. This is because, during the single-pass training, the model does not have a high chance of retraining.

5 HARDWARE ACCELERATION

NeuralHD can be accelerated in different platforms. Since there is no dependency between dimensions during the hypervector operations, FPGA is suitable hardware to parallelize the HDC in a power-efficient way. All base hypervectors are stored in Block RAM (BRAM) during encoding, which is on-chip FPGA memory. Our approach pre-fetches the weight vectors from the BRAM blocks during the encoding and stores them in the local distributed memory that can be accessed faster than BRAM. For feature vector encoding, FPGA reads the first m features of original data points ($m \leq n$). Next, it accesses the weight vector and then multiplies 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. For other encoders, the computation is all binary and perform efficiently using the lookup table (LUT) logics. Finally, the encoded hypervector can be binarized by considering the sign of the encoded data as a binary output.

6 EXPERIMENTAL RESULTS

6.1 Experimental Setup

We use in-house simulator on distributed network topologies with diverse network mediums. The simulation framework evaluates NeuralHD in a *hardware-in-the-loop* fashion. During the simulation, the simulator invokes the NeuralHD learning procedures on platforms that represent different nodes in the IoT hierarchy. HDC-based learning methods are added as the plugin module while testing data is streamed as inputs of sensing nodes. This allows us to analyze how well HDC can work with missing (lost packets in transmission) or incorrect (bit errors) data. We implement NeuralHD training and testing on three embedded platforms: Cortex A53 CPU, Kintex-7 FPGA, and Jetson Xavier GPU. For FPGA, we design the NeuralHD functionality using Verilog and synthesize it using Xilinx Vivado Design Suite [47]. The synthesis code has been implemented on the Kintex-7 FPGA KC705 Evaluation Kit. We ensure our efficiency is higher than the automated FPGA implementation at [48]. For CPU, the NeuralHD code has been written in C++ and optimized for performance. The code has been implemented on Raspberry Pi (RPi) 3B+ using ARM Cortex A53 CPU. 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 power for each connected platform while running the learning procedures. The power consumption is collected by Hioki 3337 power meter.

Table 1: 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[49, 50]
ISOLET	617	26	NA	6,238	1,559	Voice Recognition [51]
UCIHAR	561	12	NA	6,213	1,554	Activity Recognition(Mobile)[52]
FACE	608	2	NA	522,441	2,494	Face Recognition[53]
PECAN	312	3	312	22,290	5,574	Urban Electricity Prediction [54]
PAMAP2	75	5	3	611,142	101,582	Activity Recognition(IMU) [55]
APRI	36	2	3	67,017	1,241	Performance Identification[56]
PDP	60	2	5	17,385	7,334	Power Demand Prediction [57]

Table 2: Optimized DNN typologies for maximum accuracy.

Dataset	Network Topology	Dataset	Network Topology
MNIST	784, 512, 512, 10	**PECAN**	312, 512, 512, 256, 3
ISOLET	617, 256, 512, 512, 26	**PAMAP2**	75, 256, 256, 128, 128, 5
UCIHAR	561, 1024, 512, 512, 12	**APRI**	36, 256, 128, 2
FACE	607, 1024, 1024, 128, 2	**PDP**	60, 256, 256, 128, 64, 2

Table 1 summarizes the evaluated datasets. The tested benchmarks range from relatively small datasets collected in a small IoT network to large data, which includes hundreds of thousands of images of facial and non-facial data. For distributed learning, we use four datasets: (i) **PECAN** presents a dense urban area where a neighborhood may have hundreds of housing units [54]. It has 52 houses observed over the period 2014-01-1 to 2016-12-31. In each house, a set of appliances instrumented with sensors records average energy consumption. The goal is to predict the level of power consumption in the urban area. The prediction results can be used for energy management in smart cities. (ii) **PAMAP2** (Physical activity monitoring) is a dataset for human activity recognition that is widely used to understand user contexts [55]. 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 which consists of three machines [56]. 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 [57]. The goal is to identify either the high or low power state using PMC measurements of the other five servers in the cluster.

6.2 NeuralHD Accuracy

NeuralHD vs. state-of-the-art: Figure 9a compares the NeuralHD accuracy with the state-of-the-art classification algorithms, including Deep Neural Network (DNN), Support Vector Machine (SVM), and AdaBoost. The DNN models are trained with Tensorflow [58]. For each dataset, we exploit Optuna [59], the state-of-the-art hyperparameter optimization framework that finds the best DNN parameters. Table 2 lists the DNN topologies that ensure maximum classification accuracy. We exploited the Scikit-learn library [60] to evaluate other algorithms, where we used the common practice of the grid search to identify the best hyper-parameters for each model. Our evaluation shows that NeuralHD provides comparable accuracy to the sophisticated non-HDC algorithms. As compared to the existing HDC algorithms [25, 61], NeuralHD achieves, on average, 9.7% higher classification accuracy, since our

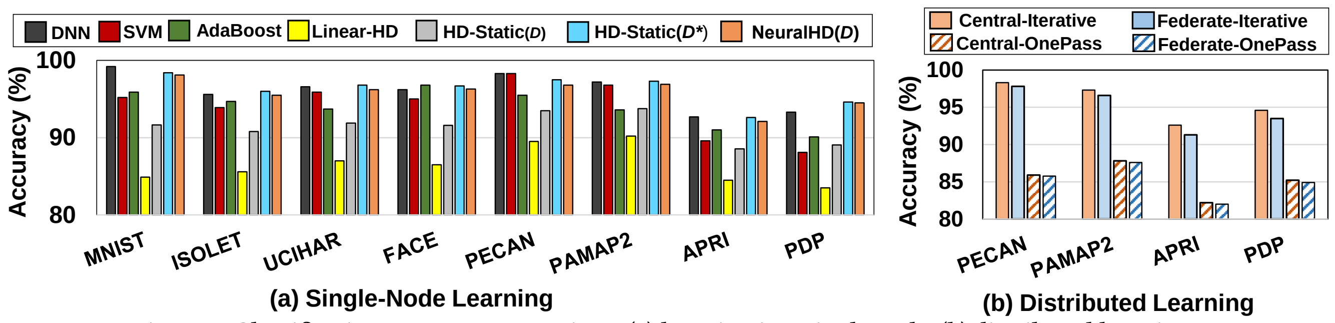


Figure 9: Classification accuracy comparison: (a) learning in a single node, (b) distributed learning

encoding method non-linearly maps data to the high-dimensional space whereas the existing HDC algorithms use a linear encoding.

NeuralHD vs. Baseline: We compare NeuralHD accuracy with the state-of-the-art HDC using linear-encoder (Linear-HD). We use NeuralHD with static encoder as a baseline that does not have the capability of regeneration (Static-HD). For the Static-HD, results are reported in two dimensionalities: (i) the same physical dimensionality ($D = 500$) as NeuralHD, (ii) the same dimensionality as NeuralHD effective dimension (D^*). We define effective dimensionality as the addition of the physical dimensions (D) with the regenerated dimensions through retraining iterations: $D^* = D + R/F \times Iter$, where R, F, and $Iter$ are regeneration rate, regeneration frequency, and the number of iterations. Our evaluation in Figure 9a shows that NeuralHD provides 4.8% higher average accuracy compared to Static-HD that uses the same number of physical dimensions. NeuralHD accuracy is comparable to Static-HD using D^* dimensions. This indicates NeuralHD capability in providing high classification accuracy while keeping the physical dimensionality low.

Centralized vs. federated learning: For datasets with multiple users, Figure 9b compares NeuralHD accuracy in four configurations: federated and centralized learning using iterative and single-pass training. NeuralHD in centralized-iterative learning provides maximum accuracy, as the cloud has access to all train data and can improve accuracy through an iterative learning process. In federated-iterative learning, NeuralHD provides very comparable accuracy to the centralized system as each edge device shares its already retrained model with the cloud. Our evaluation shows that federated-iterative learning provides, on average, only 1.1% lower accuracy than NeuralHD in centralized-iterative learning. During single-pass training, both centralized and federated learning provide comparable accuracy results, which are slightly lower than NeuralHD using iterative approaches. This is because, even in centralized learning, the cloud does not have a chance for iteratively looking at training data. The lack of retraining in single-pass reduces NeuralHD accuracy by 9.4% as compared to iterative learning.

6.3 NeuralHD vs DNN

Table 3 also compares the training/inference efficiency of NeuralHD with DNN when both algorithms run on two popular embedded platforms: Kintex-7 FPGA and Jetson Xavier. For DNN, we used network condifurations listed in Table 2. For FPGA, we used DNNWeaver V2.0 [62] for efficient implementation of the DNN inference, and FPDeep [63] for DNN training on a single FPGA device. We also used TensorFlow for GPU implementation. Both DNN implementations are optimized to maximize performance by utilizing FPGA

Table 3: NeuralHD efficiency vs. DNN running on FPGA and Xavier platforms. Absolute DNN execution time.

		Platform	MNIST	ISOLET	UCIHAR	FACE
Training	Speedup	FPGA	26.8×	16.6×	19.1×	31.7×
		Xavier	5.2×	3.3 ×	3.6×	5.7×
	Energy Improv.	FPGA	48.5×	30.4×	41.2×	61.3×
		Xavier	56.3×	34.0×	42.8×	72.9×
Inference	Speedup	FPGA	12.6×	7.9×	10.8×	17.3×
		Xavier	2.3×	1.4×	2.0×	3.1×
	Energy Improv.	FPGA	5.4×	3.7×	4.9×	6.3×
		Xavier	6.1×	4.5×	5.6×	7.3×

and GPU resources. All results listed in Table 3 are relative to DNN performance and energy efficiency.

FPGA Platform: During training, NeuralHD achieves, on average, 22.5× faster and 43.6× more energy-efficient computation compared to FPGA-based DNN implementation, respectively. The high efficiency of NeuralHD in training comes from: (i) NeuralHD capability in creating an initial model that significantly lowers the number of required retraining iterations. (ii) It eliminates the costly gradient descent for the model update. This results in a higher NeuralHD efficiency, even in terms of a single training iteration. In inference, NeuralHD provides 11.7× faster and 5.1× higher energy efficiency compared to FPGA-based DNN implementation. This higher inference efficiency comes from NeuralHD ability to reduce the number of required resources (multiply-add), on average, by 38.1× compared to the equivalent DNN. NeuralHD model compression further reduces memory footprints and provides an average 41.2× smaller model size than the DNN.

Jetson Xavier: Xavier is an ML-optimized embedded GPU architecture with 512-cores to accelerate tensor computing. As results in Table 3 reports, Xavier outperforms the FPGA implementation of DNN in terms of performance, while FPGA can still provide slightly higher energy efficiency. Our evaluation indicates that NeuralHD provides 4.2× and 49.7× (2.2× and 5.8×) faster and higher energy efficiency compared to Xavier GPU during training (inference).

NeuralHD vs. DNN in different configurations: Table 4 compares NeuralHD and DNN training efficiency using DNN with number of hidden layers and layers sizes. The results are reported when we use the same network configuration for all datasets (except the first and last layers that depend on input size and number of classes). The quality loss in Table is defined as subtraction of DNN from NeuralHD classification accuracy. The reported execution time is also normalized to NeuralHD execution time. Our result indicates that NeuralHD provides the same average accuracy to DNN with

Table 4: Comparison of different size DNN vs. NeuralHD.

# Hidden Layers	**1**		**2**		**3**		**4**	
DNN Layer Size	**256**	**512**	**256**	**512**	**256**	**512**	**256**	**512**
Quality Loss	6.4%	5.8%	3.7%	1.9%	0.6%	0%	0%	0%
Normalized Execution (Xavier)	0.53	0.62	1.1	2.3	4.7	5.9	8.3	9.12

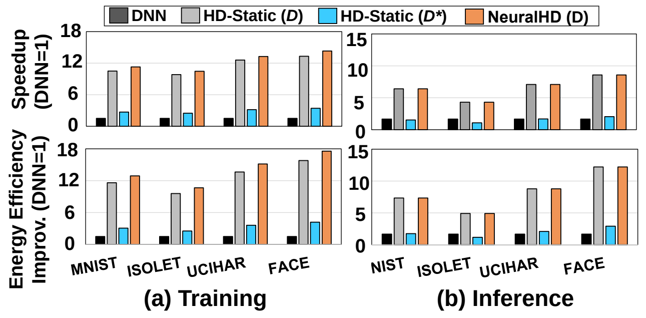


Figure 10: NeuralHD training and inference efficiency as compared to Static-HD.

about three hidden layers (with 512 size). In this configuration, DNN computation is 5.9× slower than NeuralHD in the Xavier platform.

6.4 NeuralHD CPU Efficiency

We compare NeuralHD efficiency with DNN on two embedded platforms: Raspberry Pi (RPi) 3B+ using ARM CPU. Figure 10a compares NeuralHD and the Static-HD efficiency during training and inference phases. The results are normalized to the energy and execution time of DNN running on the same ARM CPU. We use batch size equal to one for DNN, HDC, and NeuralHD. Our evaluations show that NeuralHD provides higher efficiency than DNN in both training and inference phases. These improvements are greater in the training phase, where NeuralHD not only reduces the number of training iterations but also reduces the computation complexity of a single training iteration by eliminating costly gradient computation.

In HDC-based algorithms, i.e., NeuralHD and Static-HD, the number of retraining iterations depends on the hypervector dimensionality, where a large dimensionality reduces the number of training iterations. In Static-HD, using D dimensions results in large retraining iterations, where Static-HD using D^* dimensions ($D^* > D$) can be trained with very few iterations. Although the number of NeuralHD physical dimensionality is D, its higher effective dimensionality significantly reduces the number of iterations compared to the Static-HD using D dimensions. On the other hand, the efficiency of each training iteration depends on physical dimensionality. Static-HD with using D dimensional vectors has the highest training efficiency. Although NeuralHD has D physical dimensions, the overhead of dimension regeneration increases its single-iteration cost. The Static-HD with effective dimensionality (D^*) has the lowest single-iteration efficiency, as the training needs to happen over long hypervectors ($D^* > D$). We compute total training efficiency by considering the efficiency-per-iteration along with the number of iterations. Our evaluation shows that NeuralHD provides comparable training efficiency to the Static-HD with D dimensions, while it is 3.6× and 4.2× (12.3× and 14.1×) faster and more energy-efficient than Static-HD with D^* dimensions (DNN).

Figure 10b also compares NeuralHD and the baseline HDC efficiency during the inference phase. The inference efficiency only depends on the physical dimensionality of the hypervectors. NeuralHD and the Static-HD using D physical dimensions provide the same inference efficiency. Our results also indicate that NeuralHD is 6.5× faster and 10.5× more energy-efficient than DNN, while providing comparable classification accuracy.

Centralized vs. Federated Learning: We compare the training efficiency of NeuralHD using centralized and federated learning. The results are reported in the following configurations: (i) centralized learning where edge devices are CPU (C-CPU) or FPGA (C-FPGA), and (ii) federated learning where edge devices are CPU (F-CPU) or FPGA (F-FPGA). In all configurations, the cloud is a central GPU. We experiment in all configurations when devices are supporting iterative and single-pass training. Figure 11 shows the breakdown of NeuralHD computation and communication cost during the training phase. For each application, the results are normalized to C-CPU performing the iterative training. Our evaluation shows that NeuralHD in centralized learning relies on a large amount of data movement between end node devices and the cloud. In this configuration, the communication takes a large portion of the training cost, depending on the dimensionality of the encoded data points. Using FPGA as end-node devices has a minor impact on improving NeuralHD computational efficiency, as edge devices only perform encoding locality. In this configuration, the major portion of training still performs on the cloud. Since the encoded data in both C-FPGA and C-CPU have the same size, the communication cost remains the same in both configurations.

Federated learning significantly reduces data communication by pushing a significant portion of the training task on edge devices. Our evaluation shows that NeuralHD federated learning (F-CPU) provides, on average, 1.6× faster and 1.7× higher energy efficiency than centralized learning (C-CPU). Using FPGA as edge devices result in significant improvement in total efficiency. This is because, in federated learning, edge computation is a large portion of total energy (as shown in Figure 11). For example, our experiments indicate that F-FPGA provides, on average, 1.3× faster and 1.5× energy efficiency as compared to F-CPU.

Figure 11 compares NeuralHD efficiency when cloud and edge devices are supporting single-pass training. During single-pass, NeuralHD significantly improves computation efficiency by eliminating costly iterative training. In centralized learning, this reduction in the computation cost has a minor impact on total NeuralHD efficiency, as communication dominates the total system cost. In contrast, in federated learning, edge computation takes the majority of the training cost regardless of using CPU or FPGA. Therefore, single-pass training further reduces the computing cost and makes NeuralHD closer to the real-time response. Our evaluation shows that F-FPGA in single-pass learning provides 2.6× and 3.1× (5.4× and 5.8×) faster and higher energy efficiency as compared to F-FPGA (C-FPGA) iterative training.

6.5 Regeneration Rate and Frequency

Figure 12b also shows the impact on regeneration frequency on NeuralHD final accuracy. Increasing the frequency of regeneration from $F = 1$ iteration to $F = 5$ iteration results in higher classification accuracy. As discussed in Section 3.6, a large regeneration frequency, i.e., lazy update, gives a higher chance to newly updated dimensions to increase their variance through the iterative learning

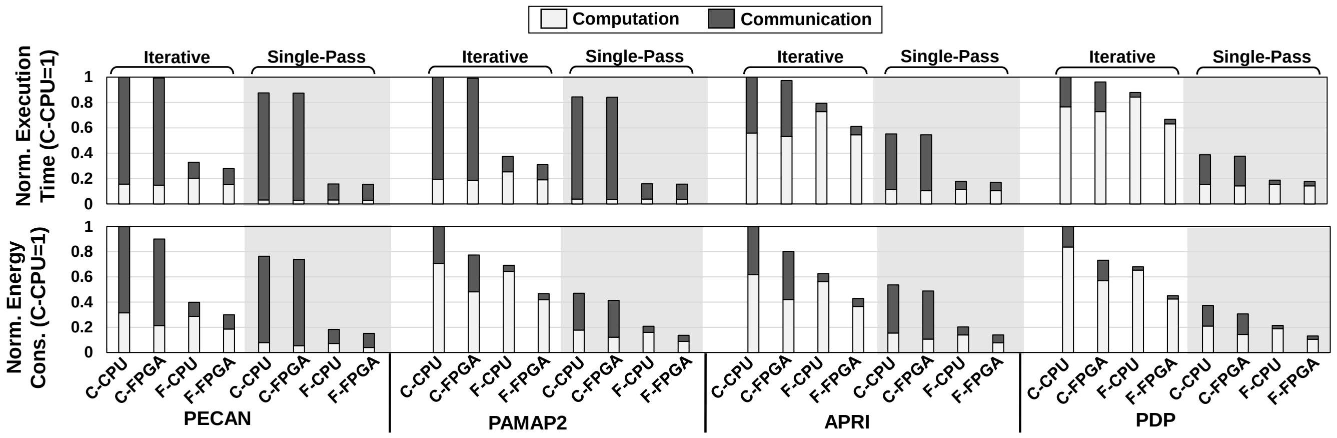


Figure 11: NeuralHD efficiency vs state-of-the-art during training and inference.

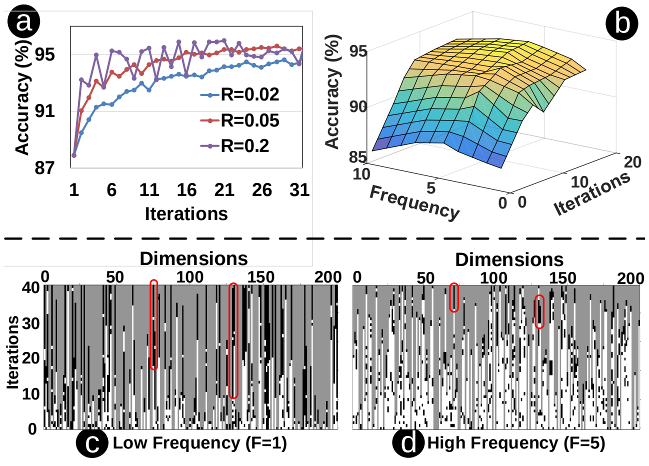


Figure 12: (a)NeuralHD accuracy using different regenerated rates (b) NeuralHD accuracy using different regeneration frequencies, (c-d) visualizing regenerated dimensions during different regeneration frequencies.

process. Therefore, they would not repeatedly select as candidate dimensions for regeneration. As Figure 12b shows, further increasing the frequency reduces NeuralHD effective dimensions resulting in lower classification accuracy. Figure 12c,d visually show NeuralHD regenerated dimensions (indices) during the learning iterations. Using high regeneration frequency, NeuralHD drops and regenerates similar dimensions in every learning iterations (Figure 12c). In contrast, using lower frequency, NeuralHD regenerates different dimensions through iterations (Figure 12d). In addition, using very large frequency results in less regeneration.

6.6 Reset vs Continuous Learning

We compare NeuralHD accuracy and number of iterations using reset and continuous learning (introduced in Section 3.4). The results are reported when both approaches are using the same physical dimension ($D = 500$) and regeneration rate. Figure 13 shows that NeuralHD with reset learning provides higher classification accuracy as compared to continuous learning, while the training takes much longer to converge. The top x-axis in Figure 13 shows the

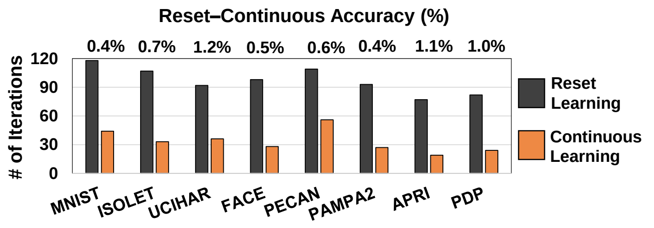


Figure 13: Reset and continuous learning: accuracy and number of training iterations.

higher accuracy values of reset learning as compared to continuous learning. In Reset learning, NeuralHD starts learning a new model in each iteration (from scratch) after updating the encoding module. In this approach, the loss of prior knowledge slows down the training process. In contrast, continuous learning continues learning from the previously trained model. Although this approach provides faster training, it may result in sub-optimal converges and potentially lower classification accuracy. For learning tasks focused on inference efficiency, Reset learning is a suitable solution as it ensures maximum accuracy. In contrast, for tasks requiring a fast training phase, continuous learning is more suitable as the learning converges with a much lower number of iterations.

6.7 Robustness to Noise

The technological and fabrication issues in highly scaled technology nodes add a significant amount of noise to both memory and computing units [64–66]. Here, we consider the impact of noise on both edge devices and the network. One of the main advantages of NeuralHD is its high robustness to noise and failure. In NeuralHD, hypervectors are random and holographic with i.i.d. components. Each hypervector stores information across all its components so that no component is more responsible for storing any information than another. This makes a hypervector robust against errors. We can exploit NeuralHD robustness to optimize total system energy by relaxing the computation and communication. We perform experiments to explore the robustness of NeuralHD and DNN to noise in both hardware and network.

Noise in Hardware: Table 5 reports the average quality loss of DNN and NeuralHD during the different percentage of hardware errors in the edge devices and the cloud. The error rates are the

Table 5: Quality loss using noisy hardware & network

Hardware Error	**1%**	**2%**	**5%**	**10%**	**15%**
DNN	3.9%	9.4%	16.3%	26.4%	40.0%
NeuralHD ($D = 2k$)	0.0%	0.0%	0.9%	3.1%	5.2%
NeuralHD ($D = 0.5k$)	0.0%	0.4%	1.4%	4.7%	7.9%
Network Error	**1%**	**20%**	**40%**	**50%**	**80%**
DNN	0%	2.3%	6.3%	14.5%	37.5%
NeuralHD ($D = 2k$)	0%	0.7%	1.3%	3.6%	6.4%
NeuralHD ($D = 0.5k$)	0.0%	1.0%	1.9%	5.6%	9.2%

percentage of random bit flips on memory storing DNN and NeuralHD models. For fairness, all DNN weights are quantized to their effective 8-bits representation. In DNN, random bit flip results in significant quality loss as corruptions on most significant bits can cause major changes in the weights. In contrast, NeuralHD has higher robustness to noise due to its redundant and holographic distribution. For example, 5% bit flip in hardware results in 16.3% quality loss in DNN, while this error results in 1.4% (0.9%) quality loss in NeuralHD using $D = 500$ ($D = 2k$). NeuralHD robustness directly depends on hypervector dimensions. As Table 5 shows, NeuralHD in higher dimensionality provides more redundant representation; thus, higher robustness to noise.

Noise in Network: We explore DNN and NeuralHD robustness to noise in the network. The noise is modeled as a loss in random packet during data communication between edge devices and the cloud. Table 5 the quality loss in the classification accuracy for DNN and NeuralHD in centralized learning. Our evaluation indicates that DNN has a much higher sensitivity to communication noise as compared to NeuralHD. In DNN, losing packets can be equivalent to losing the entire information. In contrast, NeuralHD holographic distribution and redundancy increase its robustness to the noise. In centralized learning, edge devices are transferring encoded data points to the cloud. In this configuration, an error in the network results in losing a part of the encoded hypervector. However, we observe that the cloud has a high chance of recovering the lost information during the retraining phase by replacing it with other data points with similar patterns. Our evaluations show that adding 40% network noise results in 14.5% quality loss in DNN, while this noise results in less than 3.6% (5.6%) quality loss in NeuralHD using $D = 2k$ ($D = 0.5k$) dimensions.

7 RELATED WORK

Edge-Based Learning: Several work studied the feasibility of edge-based learning as a counterpart of the centralized computing [67–69]. Prior work showed that deep neural networks computation should be split between cloud and edge devices for higher efficiency [5, 70]. These applications rely on neural network applications only limited to inference tasks. Work in [71, 72] rewrote for ML in heterogeneous hierarchical IoT systems, but they are restricted to the linearly decomposable inference computation. Recently, Google also proposed a federated learning approach [46] for collaborative learning. In their approach, each client needs to learn the local model based on the private training data to update the central model in the cloud. Prior work also investigated hot to fuse sensor data streams using ML models [73, 74]. To reduce the amount of transferred data in sensor networks, prior work studied compressive sensing techniques that statistically choose samples and features without losing too much information [75, 76]. These techniques are orthogonal to our method and can be potentially integrated with our learning solution. To summarize, our work is different from the previous work in that it enables distributed learning for both online training and inference tasks.

Hyperdimensional Computing: Prior research have applied the idea of hyperdimensional computing to diverse cognitive tasks, such as robotics [14, 21], analogy-based reasoning [77], latent semantic analysis [78], language recognition [20], clustering [79], prediction from multimodal sensor fusion [18, 19], and bio-signal processing [80, 81]. For example, the work in [78] proposed a text classification algorithm based on random indexing as a scalable alternative to latent semantic analysis. The work in [82] showed an HDC classification method for voice data. Several recent works have presented novel architecture to accelerate the HDC inference task efficiently. For example, work in [12, 23, 83–87] designed processing in-memory architecture, which supports all HDC operations inside the memory array. However, all existing HDC algorithms are using a static encoder. This makes HDC very inefficient as it requires to use of large dimensionality to solve realistic problems. In addition, the existing encoding methods do not consider non-linear interactions between features, resulting in insufficient prediction accuracy on feature vectors. To the best of our knowledge, NeuralHD is the first effort to design an adaptive encoder for HDC. Our approach identifies insignificant dimensions and regenerates them to ensure efficient and scalable learning systems.

8 CONCLUSION

In this paper, we proposed NeuralHD, a novel hyperdimensional learning system with a dynamic encoder for adaptive learning. NeuralHD identifies dimensions with less impact on the learning task and regenerates them in the encoding module to enhance learning. We show the capability of NeuralHD to support online learning from the stream of data received in real-time. Our evaluation shows that NeuralHD provides the same accuracy as state-of-the-art HD-based algorithms in much lower physical dimensionality. Therefore, NeuralHD provides, on average, 3.6× and 4.2× (12.3× and 14.1×) faster and more energy-efficient training than the HD-based algorithms (DNNs).

ACKNOWLEDGMENT

This work was supported in part by National Science Foundation (NSF) #2127780, Semiconductor Research Corporation (SRC) Task No. 2988.001, Department of the Navy, Office of Naval Research, grant #N00014-21-1-2225, and a generous gift from Cisco.

REFERENCES

[1] G. Zhu, D. Liu, Y. Du, C. You, J. Zhang, and K. Huang, "Toward an intelligent edge: wireless communication meets machine learning," *IEEE Communications Magazine*, vol. 58, no. 1, pp. 19–25, 2020.

[2] X. Wang, Y. Han, C. Wang, Q. Zhao, X. Chen, and M. Chen, "In-edge ai: Intelligentizing mobile edge computing, caching and communication by federated learning," *arXiv preprint arXiv:1809.07857*, 2018.

[3] V. Smith, C.-K. Chiang, M. Sanjabi, and A. S. Talwalkar, "Federated multi-task learning," in *Advances in Neural Information Processing Systems*, pp. 4424–4434, 2017.

[4] E. Bagdasaryan, A. Veit, Y. Hua, D. Estrin, and V. Shmatikov, "How to backdoor federated learning," *arXiv preprint arXiv:1807.00459*, 2018.

[5] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars, and L. Tang, "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," *ACM SIGPLAN Notices*, vol. 52, no. 4, pp. 615–629, 2017.

[6] J. Pan and J. McElhannon, "Future edge cloud and edge computing for internet of things applications," *IEEE Internet of Things Journal*, vol. 5, no. 1, pp. 439–449, 2017.

[7] H. Li, K. Ota, and M. Dong, "Learning iot in edge: Deep learning for the internet of things with edge computing," *IEEE network*, vol. 32, no. 1, pp. 96–101, 2018.

[8] O. Yilmaz, "Symbolic computation using cellular automata-based hyperdimensional computing," *Neural computation*, vol. 27, no. 12, pp. 2661–2692, 2015.

[9] 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.

[10] P. Kanerva, J. Kristofersson, and A. Holst, "Random indexing of text samples for latent semantic analysis," in *Proceedings of the 22nd annual conference of the cognitive science society*, vol. 1036, Citeseer, 2000.

[11] P. Poduval *et al.*, "Stochd: Stochastic hyperdimensional system for efficient and robust learning from raw data," in *IEEE/ACM Design Automation Conference (DAC)*, 2021.

[12] 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.

[13] T. F. Wu, H. Li, P.-C. Huang, A. Rahimi, J. M. Rabaey, H.-S. P. Wong, M. M. Shulaker, and S. Mitra, "Brain-inspired computing exploiting carbon nanotube fets and resistive ram: Hyperdimensional computing case study," in *2018 IEEE International Solid-State Circuits Conference-(ISSCC)*, pp. 492–494, IEEE, 2018.

[14] A. Mitrokhin, P. Sutor, C. Fermüller, and Y. Aloimonos, "Learning sensorimotor control with neuromorphic sensors: Toward hyperdimensional active perception," *Science Robotics*, vol. 4, no. 30, p. eaaw6736, 2019.

[15] A. Hérnandez-Cano *et al.*, "Reghd: Robust and efficient regression in hyperdimensional learning system," in *IEEE/ACM Design Automation Conference (DAC)*, 2021.

[16] P. Poduval *et al.*, "Cognitive correlative encoding for genome sequence matching in hyperdimensional system," in *IEEE/ACM Design Automation Conference (DAC)*, 2021.

[17] A. Mitrokhin, P. Sutor, D. Summers-Stay, C. Fermüller, and Y. Aloimonos, "Symbolic representation and learning with hyperdimensional computing,"

[18] O. Räsänen and S. Kakouros, "Modeling dependencies in multiple parallel data streams with hyperdimensional computing," *IEEE Signal Processing Letters*, vol. 21, no. 7, pp. 899–903, 2014.

[19] O. Rasanen and J. Saarinen, "Sequence prediction with sparse distributed hyperdimensional coding applied to the analysis of mobile phone use patterns," *IEEE Transactions on Neural Networks and Learning Systems*, vol. PP, no. 99, pp. 1–12, 2015.

[20] A. Joshi, J. Halseth, and P. Kanerva, "Language geometry using random indexing," *Quantum Interaction 2016 Conference Proceedings*, In press.

[21] S. Jockel, "Crossmodal learning and prediction of autobiographical episodic experiences using a sparse distributed memory," 2010.

[22] L. Ge and K. K. Parhi, "Classification using hyperdimensional computing: A review," *arXiv preprint arXiv:2004.11204*, 2020.

[23] H. Li, T. F. Wu, A. Rahimi, K.-S. Li, M. Rusch, C.-H. Lin, J.-L. Hsu, M. M. Sabry, S. B. Eryilmaz, J. Sohn, *et al.*, "Hyperdimensional computing with 3d vrram in-memory kernels: Device-architecture co-design for energy-efficient, error-resilient language recognition," in *Electron Devices Meeting (IEDM), 2016 IEEE International*, pp. 16–1, IEEE, 2016.

[24] P. Poduval *et al.*, "Hyperdimensional self-learning systems robust to technology noise and bit-flip attacks," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, IEEE, 2021.

[25] M. Imani, Y. Kim, S. Riazi, J. Messerly, P. Liu, F. Koushanfar, and T. Rosing, "A framework for collaborative learning in secure high-dimensional space," in *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*, pp. 435–446, IEEE, 2019.

[26] A. Hérnandez-Cano *et al.*, "Prid: Model inversion privacy attacks in hyperdimensional learning systems," in *IEEE/ACM Design Automation Conference (DAC)*, 2021.

[27] A. Rahimi, P. Kanerva, and J. M. Rabaey, "A robust and energy-efficient classifier using brain-inspired hyperdimensional computing," in *Proceedings of the 2016 International Symposium on Low Power Electronics and Design*, pp. 64–69, 2016.

[28] M. Imani, Z. Zou, S. Bosch, S. A. Rao, S. Salamat, V. Kumar, Y. Kim, and T. Rosing, "Revisiting hyperdimensional learning for fpga and low-power architectures," in *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pp. 221–234, IEEE, 2021.

[29] M. Imani, J. Morris, J. Messerly, H. Shu, Y. Deng, and T. Rosing, "Bric: Locality-based encoding for energy-efficient brain-inspired hyperdimensional computing," in *Proceedings of the 56th Annual Design Automation Conference 2019*, pp. 1–6, 2019.

[30] J. H. Morrison and P. R. Hof, "Life and death of neurons in the aging brain," *Science*, vol. 278, no. 5337, pp. 412–419, 1997.

[31] E. I. Rugarli and T. Langer, "Mitochondrial quality control: a matter of life and death for neurons," *The EMBO journal*, vol. 31, no. 6, pp. 1336–1349, 2012.

[32] F. H. Gage and S. Temple, "Neural stem cells: generating and regenerating the brain," *Neuron*, vol. 80, no. 3, pp. 588–601, 2013.

[33] L. Gao, W. Guan, M. Wang, H. Wang, J. Yu, Q. Liu, B. Qiu, Y. Yu, Y. Ping, X. Bian, *et al.*, "Direct generation of human neuronal cells from adult astrocytes by small molecules," *Stem cell reports*, vol. 8, no. 3, pp. 538–547, 2017.

[34] G. Stoll and H. W. Müller, "Nerve injury, axonal degeneration and neural regeneration: basic insights," *Brain pathology*, vol. 9, no. 2, pp. 313–325, 1999.

[35] "Number of new generated neurons every day, Nicolas Toni." https://wp.unil.ch/discoverunil/2017/06/we-create-1500-new-neurons-every-day/.

[36] S. Ackerman *et al.*, *Discovering the brain.* National Academies Press, 1992.

[37] Z. Zou *et al.*, "Manihd: Efficient hyper-dimensional learning using manifold trainable encoder," in *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 850–855, IEEE, 2021.

[38] M. Imani, S. Bosch, M. Javaheripi, B. Rouhani, X. Wu, F. Koushanfar, and T. Rosing, "Semihd: Semi-supervised learning using hyperdimensional computing," in *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 1–8, IEEE, 2019.

[39] P. Kanerva, "Encoding structure in boolean space," in *ICANN 98*, pp. 387–392, Springer, 1998.

[40] B. Pakkenberg, D. Pelvig, L. Marner, M. J. Bundgaard, H. J. G. Gundersen, J. R. Nyengaard, and L. Regeur, "Aging and the human neocortex," *Experimental gerontology*, vol. 38, no. 1-2, pp. 95–99, 2003.

[41] B. B. Andersen, H. J. G. Gundersen, and B. Pakkenberg, "Aging of the human cerebellum: a stereological study," *Journal of Comparative Neurology*, vol. 466, no. 3, pp. 356–365, 2003.

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

[43] B. Scholkopf, K.-K. Sung, C. J. Burges, F. Girosi, P. Niyogi, T. Poggio, and V. Vapnik, "Comparing support vector machines with gaussian kernels to radial basis function classifiers," *IEEE transactions on Signal Processing*, vol. 45, no. 11, pp. 2758–2765, 1997.

[44] L. Marner, J. R. Nyengaard, Y. Tang, and B. Pakkenberg, "Marked loss of myelinated nerve fibers in the human brain with age," *Journal of comparative neurology*, vol. 462, no. 2, pp. 144–152, 2003.

[45] M. F. Paredes, S. F. Sorrells, A. Cebrian-Silla, K. Sandoval, D. Qi, K. W. Kelley, D. James, S. Mayer, J. Chang, K. I. Auguste, *et al.*, "Does adult neurogenesis persist in the human hippocampus?," *Cell Stem Cell*, vol. 23, no. 6, pp. 780–781, 2018.

[46] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth, "Practical secure aggregation for privacy-preserving machine learning," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pp. 1175–1191, 2017.

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

[48] S. Salamat, M. Imani, B. Khaleghi, and T. Rosing, "F5-hd: Fast flexible fpga-based framework for refreshing hyperdimensional computing," in *FPGA*, pp. 53–62, ACM, 2019.

[49] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[50] D. Ciregan, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for image classification," in *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pp. 3642–3649, IEEE, 2012.

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

[52] D. Anguita, A. Ghio, L. Oneto, X. Parra, and J. L. Reyes-Ortiz, "Human activity recognition on smartphones using a multiclass hardware-friendly support vector machine," in *International workshop on ambient assisted living*, pp. 216–223, Springer, 2012.

[53] A. Angelova, Y. Abu-Mostafam, and P. Perona, "Pruning training sets for learning of object categories," in *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, IEEE, 2005.

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

[55] A. Reiss and D. Stricker, "Introducing a new benchmarked dataset for activity monitoring," in *Wearable Computers (ISWC), 2012 16th International Symposium on*, pp. 108–109, IEEE, 2012.

[56] M. Zaharia, R. S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataraman, M. J. Franklin, *et al.*, "Apache spark: a unified engine for big data processing," *Communications of the ACM*, vol. 59, no. 11, pp. 56–65, 2016.

[57] Y. Kim, P. Mercati, A. More, E. Shriver, and T. Rosing, "P4: Phase-based power/performance prediction of heterogeneous systems via neural networks," in *Computer-Aided Design (ICCAD), 2017 IEEE/ACM International Conference on*, pp. 683–690, IEEE, 2017.

[58] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *arXiv preprint arXiv:1603.04467*, 2016.

[59] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," in *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 2623–2631, 2019.

[60] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, *et al.*, "Scikit-learn: Machine learning in python," *Journal of Machine Learning Research*, vol. 12, no. Oct, pp. 2825–2830, 2011.

[61] M. Imani, J. Messerly, F. Wu, W. Pi, and T. Rosing, "A binary learning framework for hyperdimensional computing," in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 126–131, IEEE, 2019.

[62] H. Sharma, J. Park, D. Mahajan, E. Amaro, J. K. Kim, C. Shao, A. Mishra, and H. Esmaeilzadeh, "From high-level deep neural models to fpgas," in *Microarchitecture (MICRO), 2016 49th Annual IEEE/ACM International Symposium on*, pp. 1–12, IEEE, 2016.

[63] T. Geng, T. Wang, A. Sanaullah, C. Yang, R. Xu, R. Patel, and M. Herbordt, "Fpdeep: Acceleration and load balancing of cnn training on fpga clusters," in *2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pp. 81–84, IEEE, 2018.

[64] S.-H. Lee, "Technology scaling challenges and opportunities of memory devices," in *2016 IEEE International Electron Devices Meeting (IEDM)*, pp. 1–1, IEEE, 2016.

[65] K. T. Lee, W. Kang, E.-A. Chung, G. Kim, H. Shim, H. Lee, H. Kim, M. Choe, N.-I. Lee, A. Patel, *et al.*, "Technology scaling on high-k & metal-gate finfet bti reliability," in *2013 IEEE International Reliability Physics Symposium (IRPS)*, pp. 2D–1, IEEE, 2013.

[66] H. Esmaeilzadeh, E. Blem, R. S. Amant, K. Sankaralingam, and D. Burger, "Dark silicon and the end of multicore scaling," in *Computer Architecture (ISCA), 2011 38th Annual International Symposium on*, pp. 365–376, IEEE, 2011.

[67] 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.

[68] L. Tong, Y. Li, and W. Gao, "A hierarchical edge cloud architecture for mobile computing," in *INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications, IEEE*, pp. 1–9, IEEE, 2016.

[69] 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.

[70] J. H. Ko, T. Na, M. F. Amir, and S. Mukhopadhyay, "Edge-host partitioning of deep neural networks with feature space encoding for resource-constrained internet-of-things platforms," *arXiv preprint arXiv:1802.03835*, 2018.

[71] J. Venkatesh, C. Chan, A. S. Akyurek, and T. S. Rosing, "A modular approach to context-aware iot applications," in *Internet-of-Things Design and Implementation (IoTDI), 2016 IEEE First International Conference on*, pp. 235–240, IEEE, 2016.

[72] H. Grunert and A. Heuer, "Rewriting complex queries from cloud to fog under capability constraints to protect the users' privacy," *Open Journal of Internet Of Things (OJIOT)*, vol. 3, no. 1, pp. 31–45, 2017.

[73] "Aws greengrass." https://aws.amazon.com/greengrass/.

[74] B. Khaleghi, A. Khamis, F. O. Karray, and S. N. Razavi, "Multisensor data fusion: A review of the state-of-the-art," *Information fusion*, vol. 14, no. 1, pp. 28–44, 2013.

[75] S. Qaisar, R. M. Bilal, W. Iqbal, M. Naureen, and S. Lee, "Compressive sensing: From theory to applications, a survey," *Journal of Communications and networks*, vol. 15, no. 5, pp. 443–456, 2013.

[76] S. Shalev-Shwartz and S. Ben-David, *Understanding machine learning: From theory to algorithms.* Cambridge university press, 2014.

[77] P. Kanerva, "What we mean when we say "what's the dollar of mexico?": Prototypes and mapping in concept space," in *AAAI Fall Symposium: Quantum Informatics for Cognitive, Social, and Semantic Processes*, pp. 2–6, 2010.

[78] P. Kanerva, J. Kristofersson, and A. Holst, "Random indexing of text samples for latent semantic analysis," in *Proceedings of the 22nd annual conference of the cognitive science society*, vol. 1036, Citeseer, 2000.

[79] A. Hernández-Cano *et al.*, "A framework for efficient and binary clustering in high-dimensional space," in *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1859–1864, IEEE, 2021.

[80] A. Burrello, K. Schindler, L. Benini, and A. Rahimi, "One-shot learning for ieeg seizure detection using end-to-end binary operations: Local binary patterns with hyperdimensional computing," in *2018 IEEE Biomedical Circuits and Systems Conference (BioCAS)*, pp. 1–4, IEEE, 2018.

[81] D. Kleyko, A. Rahimi, D. A. Rachkovskij, E. Osipov, and J. M. Rabaey, "Classification and recall with binary hyperdimensional computing: Tradeoffs in choice of density and mapping characteristics," *IEEE transactions on neural networks and learning systems*, no. 99, pp. 1–19, 2018.

[82] M. Imani, D. Kong, A. Rahimi, and T. Rosing, "Voicehd: Hyperdimensional computing for efficient speech recognition," in *2017 IEEE international conference on rebooting computing (ICRC)*, pp. 1–8, IEEE, 2017.

[83] 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.

[84] T. Wu, P. Huang, A. Rahimi, H. Li, J. Rabaey, P. Wong, and S. Mitra, "Brain-inspired computing exploiting carbon nanotube fets and resistive ram: Hyperdimensional computing case study," in *IEEE Intl. Solid-State Circuits Conference (ISSCC)*, IEEE, 2018.

[85] M. Imani, S. Pampana, S. Gupta, M. Zhou, Y. Kim, and T. Rosing, "Dual: Acceleration of clustering algorithms using digital-based processing in-memory," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 356–371, IEEE, 2020.

[86] A. Kazemi *et al.*, "Mimhd: Accurate and efficient hyperdimensional inference using multi-bit in-memory computing," in *2021 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, pp. 1–6, IEEE, 2021.

[87] 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*, vol. 39, no. 10, pp. 2422–2433, 2019.

Appendix: Artifact Description/Artifact Evaluation

SUMMARY OF THE EXPERIMENTS REPORTED

We ran the NeuralHD learning algorithm that's described in the paper with CPU. The following artifact contains the optimized code as well as the code for the evaluation/visualization as detailed in the paper in CPU, implemented in python.

Author-Created or Modified Artifacts:

```
Persistent ID: https://gitlab.com/biaslab/neuralhd
Artifact name: Source Code for NeuralHD
```

BASELINE EXPERIMENTAL SETUP, AND MODIFICATIONS MADE FOR THE PAPER

Relevant hardware details: Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz 2.80 GHz

Operating systems and versions: Windows 10 Home, version 20H2, build 19042.1083

Compilers and versions: Python 3.7.8

Applications and versions: N/A

Libraries and versions: scikit-learn 0.23.2

Key algorithms: Hyperdimensional Classification

Input datasets and versions: MNIST, smart home