Federated Hyperdimensional Computing: Comprehensive Analysis and Robust Communication

YE TIAN*, RISHIKANTH CHANDRASEKARAN*, KAZIM ERGUN*, XIAOFAN YU, and TAJANA ROSING†, Department of Computer Science and Engineering, University of California San Diego, USA

Federated learning is a distributed learning method by training the model in locally multiple clients, which has been used in numerous fields. Current convolutional neural networks (CNN)-based federated learning approaches face challenges from computational cost, communication efficiency and robust communication. Recently, Hyper Dimensional Computing (HDC) has been recognized as a promising technique to address these challenges. HDC encodes data as high-dimensional vectors and enables lightweight training and communication through simple parallel vector operations. Several HDC-based federated learning methods have been proposed. Although existing methods reduce computational efficiency and communication cost, they are difficult to handle complex learning tasks and are not robust to unreliable wireless channels. In this work, we innovatively introduce a synergetic federated learning framework, FHDnn. With advantage of the complementary strengths of CNN and HDC, FHDnn can achieve optimal performance on complex image tasks while maintaining good computational and communication efficiency. Secondly, we demonstrate in detail the convergence of using HDC in a generalized federated learning framework, providing theoretical guarantees for HDC-based federated learning approach. Finally, we design 3 communication strategies to further improve the communication efficiency of FHDnn by 32×. Experiments demonstrate that FHDnn converges 3× faster than CNN-based federated learning methods, reduces the communication cost by 2112×, the local computation and energy consumption by 192×. In addition, it has good robustness to unreliable communication with bit errors, noise and packet loss.

CCS Concepts: • **Computing methodologies → Distributed computing methodologies**.

Additional Key Words and Phrases: Federated Learning, Hyperdimensional Computing, Convergence Analysis and Proof

ACM Reference Format:
Ye Tian, Rishikanth Chandrasekaran, Kazim Ergun, Xiaofan Yu, and Tajana Rosing. 2025. Federated Hyperdimensional Computing: Comprehensive Analysis and Robust Communication. 1, 1 (March 2025), 30 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Recent years have witnessed unprecedented growth in IoT data collection, with an estimated 40 billion interconnected devices generating over 79 zettabytes of data by 2025 [22]. This massive

*These authors contributed equally to this research.
†Tajana Rosing is corresponding author.

Authors' address: Ye Tian, yet002@ucsd.edu; Rishikanth Chandrasekaran, r3chandr@ucsd.edu; Kazim Ergun, kergun@ucsd.edu; Xiaofan Yu, x1yu@ucsd.edu; Tajana Rosing, tajana@ucsd.edu, Department of Computer Science and Engineering, University of California San Diego, USA.

data enables advanced deep learning applications across various domains [38]. In traditional cloud-centric deep learning frameworks, data are usually collected by remote clients and centrally stored in servers or data centers with powerful computing capabilities for model training. However, clients may be reluctant to share data with servers due to privacy concerns. In addition, transferring huge datasets between clients and servers can place a heavy burden on limited network resources. In recent years, federated learning (FL) [41], as an innovative alternative to centralized learning has attracted significant attention. FL supports machine learning in edge networks without data sharing. Each client trains a model independently based on its local dataset and uploads the model parameters to a centralized server. Subsequently, the server aggregates the model parameters from all participating clients to form a unified global model.

Existing FL methods face three main challenges: computational cost, communication efficiency, and robust communication. Convolutional neural network (CNN)-based federated learning methods demand significant computational and communication resources due to their large number of parameters. For instance, training a ResNet-18 model [18] in a federated setting with one server and 100 clients requires 110 GB of data transfer and 18,000 GFLOPS of computation after 100 training rounds. To address these challenges, prior studies mainly uses the quantization model [49] and some pruning methods [26] to reduce the amount of computation. Approaches to address communication bottlenecks have focused on reducing the size of updates through the model [34] and reducing the number of communication rounds[49]. Last but not least, unreliable wireless channel can interfere with the proper transmission of signals due to noise, bit errors, and packet loss, resulting in lower model accuracy. A common approach to solving this problem is to use multiple-access techniques [17] (e.g., TDMA, OFDMA) to prevent interference and error-correcting codes to overcome the noise. If errors persist, transmission failures need to be detected and recovered through acknowledgement, retransmission and timeout mechanisms [37]. However, this is very expensive. Typically, a large amount of wireless resources are required to achieve error-free communication, which further increases energy consumption and limits communication efficiency, thereby reducing the training speed and convergence of FL. Existing methods are attempting to address these challenges, but addressing all of these issues effectively and simultaneously remains difficult [59].

Recently, several studies have shown that Hyperdimensional Computing (HDC) [16] is a very effective way to address the bottleneck of FL. HDC, a computational paradigm that encodes data into high-dimensional vectors to simulate brain-inspired computations, has been widely used in several learning domains, including bio-signal processing [47], activity classification [32], speech recognition [25], and multimodal sensor fusion [66]. HDC is much more computationally and communicatively efficient than traditional deep learning methods since it enables fast and lightweight learning through simple operations, and thus is well suited to run on resource-constrained edge devices. Some federal learning frameworks based on HDC have been proposed [21, 24, 40, 64], however, they are difficult to scale to complex tasks, have no theoretical guarantees, and are not good enough in terms of communication robustness. For example, most of exciting HDC-based federated learning methods have only been validated on datasets like MNIST [11], ISOLET [1], and some simple HAR datasets [2], but struggle to handle more complex image classification tasks. The state-of-the-art (SOTA) method [63] also achieves only 49% best accuracy on CIFAR-10 [36]. Therefore, in this work, our main goal is to improve the accuracy of the HDC-based federated learning framework on complex tasks, taking into account its optimal computational cost, communication efficiency, and robust communication.

In this paper, we propose FHDnn, a synergetic federated learning framework combining pre-trained CNN to learn complex feature structures and HDC for computational and communication efficiency. FHDnn encodes the extracted features as hypervectors and trains the HD classifier in a horizontally federated manner [62], each client locally trains a model based on the data it possesses, which shares the same features. Subsequently, the central server communicates with all clients and aggregates the model parameters to update a unified global model. Building on our prior work [6], this paper provides a systematic and comprehensive extension with several significant contributions. **First**, in this paper, we comprehensively analyze the training process of HDC from both statistical and optimization perspectives, and formalize the training algorithm of HDC in a generic federated learning scenario. **Second**, for the first time, we present a complete proof of the convergence property of HDC within a general federated learning framework, providing a critical theoretical guarantee for HDC-based federated learning. **In addition**, we test the performance of the defined HDC training algorithm on five datasets. The experiments verify the convergence speed and performance of the HDC training algorithm in a generalized federated learning scenario, which is consistent with our proofs and analysis. **Last but not least**, we design 3 novel communication strategies to further improve the communication efficiency of FHDnn, which increases its communication efficiency 32× under the condition of losing only 2.9% accuracy.

In summary, our main contributions are as follows:

(1) In this paper, we introduce FHDnn, a new synergetic federated learning framework combining CNN and HDC in more detail. It combines the advantages of both CNN and HDC, and can achieve significant accuracy improvement in complex image classification while maintaining high computational and communication efficiency.

(2) For the first time, we provide a complete proof in a general federated learning framework that HDC can converge at a rate of $O(\frac{1}{T})$ and analyze several of its properties, where T denotes the number of communication rounds. This provides important theoretical guarantees for HDC-based federated learning research.

(3) We tolerate the occurrence of errors during transmission and analyze the performance of FHDnn under three common unreliable network channel in FL: packet loss, noise injection, and bit error. The experiments show that the proposed method can well tolerate perturbations in the client model.

(4) We propose three strategies to improve communication efficiency and comprehensively validate FHDnn on different benchmark datasets. The experiment shows that with the optimization of the communication strategy, the convergence speed of FHDnn is 3× faster than that of CNN, the communication cost is reduced by 2112×, and the local computation and energy consumption are reduced by 192×.

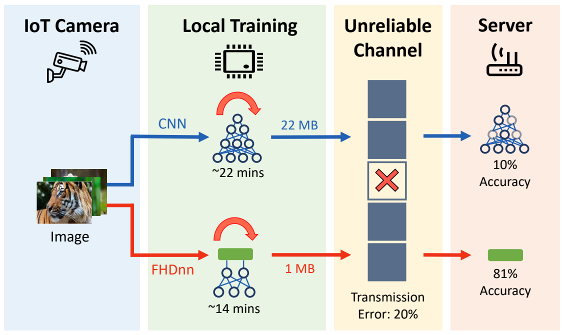


Fig. 1. FHDnn against CNNs for federated learning

2 RELATED WORK

In this section, we review the efforts made by existing methods to improve the computational cost and communication efficiency of federated learning, and summarize the federal learning frameworks based on HDC.

2.1 Improving Computation Efficiency in FL

To overcome the challenge of limited client resources, much work has explored low-complexity neural network architectures and lightweight algorithms suitable for edge devices, e.g., pruning [26] and using quantized models [49], which are also helpful for reducing computation. A small subset of the proposed approaches specifically devotes their attention to resolving the computational issues in federated learning. In [61], a "soft-training" method was introduced to dynamically compress the original training model into a smaller restricted volume through rotating parameter training. In each round, it lets different parts of the model parameters alternately join the training, but maintains the complete model for federated aggregation. The authors of [55] suggested dividing the model into sub-models, then using only a few sub-models for partial federated training while keeping the rest of the parameters fixed. During training, sub-model capacities are gradually increased until they reach the full model. Along similar lines, federated dropout [5] is a technique that enables each client to locally operate on a smaller sub-model while still providing updates that can be applied to the larger global model on the server. Finally, the technique presented in [53], called splitfed learning, combines the strengths of federated learning and split learning by splitting a NN into client-side and server-side sub-networks during federated training.

2.2 Improving Communication Efficiency in FL

FedAvg [44] improved the communication efficiency of federated learning framework by allowing for the clients to run multiple local SGD steps per communication round. Another approach that directly affects local training is to modify model complexity. Some examples are pruning [67], restricting the model weights to be numbers at a certain bitwidth [10], and bounding the model size [45]. Some work explored methods to reduce the communicated model (or gradient) size without altering the original local models. They typically perform a form of compression, that is, instead of transmitting the raw model/gradient data, one transmits a compressed representation with fewer bits, for instance by means of limiting bitwidth (quantization) or enforcing sparsity (sparsification). In particular, a popular class of quantization operators is based on random dithering [19]. Sparsification methods decrease the number of non-zero entries in the communicated data to obtain sparse vectors [57]. Structured and sketched updates are also proposed in [33], which can be further supported by lossy compression and federated dropout [5]. Some other approaches include randomized techniques such as stochastic rounding [52], subsampling [34], and randomized approximation [35]. In addition, some work has tried simple client-selection heuristics, such as selecting clients with higher losses [3], sampling clients of larger update norm with higher probability [8], and sampling clients with probabilities proportional to their local dataset size [44].

2.3 HDC-based Federated Learning

The lightweight nature of the HDC model makes it well-suited to run on resource-constrained edge devices. SecureHD [24] first proposes a federated learning framework incorporating HDC. FedHD [64] demonstrates how to deploy HDC's federated learning framework on real edge computing devices and under realistic network conditions. FL-HDC [21] polarizes the parameters of the model and proposes a retraining mechanism with adaptive learning rate to improve communication efficiency. Few-shot federated learning [50] proposes a novel compression method to optimize communication efficiency. HyperFeel [40] designs an online update mechanism in the local client, which can achieve lower communication overhead. MultimodalHD [65] feeds multimodal hypervectors to the focus fusion module to learn richer multimodal representations. Some works also studied security issues in HD-based federated learning frameworks [30]. Although these methods have improved computational and communication efficiency, they only perform well on MNIST [11], ISOLET [1], and some simple HAR datasets [2], but struggle to handle more complex image datasets. The best accuracies on CIFAR-10 [36] is only 49.2% [63]. In contrast, FHDnn achieves over 80% accuracy on CIFAR-10 in less than 25 rounds of communication and is more robust against unreliable channels.

3 BACKGROUND ON HYPERDIMENSIONAL COMPUTING

In this section, we introduce the encoding, learning, and training processes of HDC. We first illustrate the encoding method of HDC for mapping input signals into hypervectors, and then discuss the principles and process of its learning. Finally, we focus on analyzing its training process, which is important for further extending its application in federated learning scenarios and to analyze its properties.

3.1 Encoding of Hyperdimensional Computing

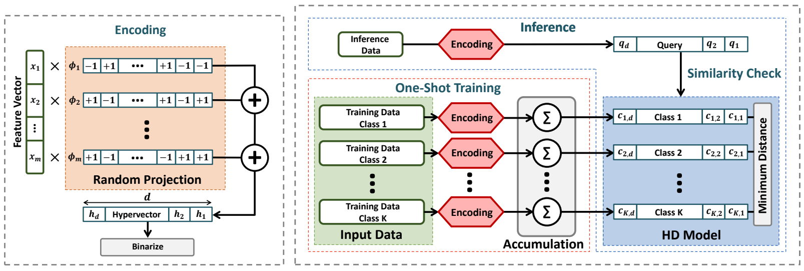
HDC performs cognitive tasks using high-dimensional vectors, also known as hypervectors. These hypervectors typically range from 1,000 to 10,000 dimensions and are composed of independent and identically distributed (i.i.d.) components. Due to the randomness and high dimensionality, any two randomly chosen points in hyperdimensional space are nearly orthogonal [28].

The first step of HDC is to map/encode the input signal (e.g., an image, feature vector, or a time-series window) into hypervectors. This encoding process is consistent across various HDC applications. Assume the input $\mathbf{x} \in \mathcal{X}$ is represented by the vector $\mathbf{x} = [x_1, x_2, ..., x_m]^T$, where x_is denote the features and m is the length of input vector. HDC encoding operation maps the input data to its high-dimensional representation $\mathbf{h} \in \mathcal{H}$ with dimension $d \gg m$ under some function $\phi : \mathcal{X} \rightarrow \mathcal{H}$. Some encoding algorithms have been proposed for different memory-compute trade-offs, such as base-level (a.k.a position-ID) [25], permutation [48], and random projection [23].

In this paper, we refer to the random projection encoding [23], but our methodology can be extended to any other encoding approach. Random projection encoding embeds the data into a high-dimensional Euclidean space under a random linear map. The output of this mapping can be quantized with minimal loss of information for better computational efficiency. If quantized, the HD embedding is constructed as $\phi(\mathbf{x}) = \text{sign}(\mathbf{\Phi x})$ under the encoding function $\phi : \mathbb{R}^m \rightarrow \mathbb{Z}^d$, the rows of which $\mathbf{\Phi} \in \mathbb{R}^{d \times m}$ are generated by randomly sampling directions from the m-dimensional unit sphere. Here, $\text{sign}(\mathbf{\Phi x})$ is the element-wise sign function returning +1 if $\mathbf{\Phi x} \geq 0$ and -1 otherwise. Fig. 2(a) shows an overview of HDC encoding.

3.2 Learning of Hyperdimensional Computing

Many learning tasks can be implemented in the HD domain. Here, we focus on classification, one of the most common learning problems. Consider a dataset $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{n}$, where $\mathbf{x_i} \in \mathcal{X} \subset \mathbb{R}^m$ represents the input samples, and $y_i \in C$ indicates their class labels. For HD learning, we first encode the entire set of data samples in $\mathcal{D}$ into hyperdimensional vectors such that $\mathbf{h}_i = \phi(\mathbf{x}_i)$ is a hypervector in the d-dimensional inner-product space $\mathcal{H}$. These high-dimensional embeddings represent data in a way that admits linear learning algorithms, even if the data was not separable to begin with. The common approach to learning with HD representations involves *bundling* training examples for each class into a set of "prototypes," which are subsequently used for classification. The bundling operator is used to compile a set of elements in $\mathcal{H}$ and is defined as a function



(a) HDC encoding (b) HDC classification

Fig. 2. Hyperdimensional learning overview

$\oplus : \mathcal{H} \times \mathcal{H} \rightarrow \mathcal{H}$. The function takes two points in $\mathcal{H}$ and returns a third point similar to both operands. We bundle all the encoded hypervectors that belong to the k-th class to construct the corresponding prototype $\mathbf{c}_k$:

$$\mathbf{c}_k = \bigoplus_{i \text{ s.t. } y_i = k} \mathbf{h}_i \tag{1}$$

Given a query data $\mathbf{x}_q \in \mathcal{X}$ for which we search for the correct label to classify, we take the encoded hypervector $\mathbf{h}_q \in \mathcal{H}$ and return the label of the most similar prototype:

$$\hat{y}_q = k^* = \underset{k \in 1,\ldots,K}{\operatorname{argmax}}\ \delta(\mathbf{h}_q, \mathbf{c}_k) \tag{2}$$

where δ is a similarity metric. The detailed process is as follows: *1). One-Shot Training*. The bundling operator $\oplus$ is often chosen to be element-wise sum. In this case, the class prototypes are obtained by adding all hypervectors with the same class label. Then, the operation in Equation (1) is simply calculated as: $\mathbf{c}_k = \sum_{i \text{ s.t. } y_i = k} \mathbf{h}_i$. It can be considered a single-pass training , as the entire dataset is used only once—without iterations—to construct the model. *2).Inference.* The similarity metric δ is typically defined as cosine similarity, which measures the angle between two vectors in an inner product space. Using cosine similarity, Equation (2) can be expressed in terms of the dot product and vector magnitudes as follows:

$$\hat{y}_q = k^* = \underset{k \in 1,\ldots,K}{\operatorname{argmax}} \frac{\langle \mathbf{c}_k, \mathbf{h}_q \rangle}{\|\mathbf{c}_k\|} \tag{3}$$

3). Retraining. One-shot training often does not result in sufficient accuracy for complex tasks. A common approach is to fine-tune the class prototypes using a few iterations of retraining [25, 27, 31, 48]. We use the perceptron algorithm [14] to update the class hypervectors for mistpredicted samples. The model is updated only if the query in Equation (3) returns an incorrect label. Let $y_q = k$ and $\hat{y}_q = k'$ be the correct and mispredicted labels respectively. So, the new class prototypes after the retraining iteration are: $\mathbf{c}_k = \mathbf{c}_k + \alpha \mathbf{h}_q, \quad \mathbf{c}_{k'} = \mathbf{c}_{k'} - \alpha \mathbf{h}_q$, where α is the HD learning rate, controlling the amount of change we make to the model during each iteration. Fig. 2(b) shows an overview of HDC for classification.

3.3 Training Analysis of Hyperdimensional Computing

When addressing specific tasks, the training of HDC can be categorized into single-pass and iterative training. The single-pass training process and the dot-product-based inference method are highly analogous to Fisher's Linear Discriminant Analysis [13]. If the target accuracy is not achieved, we need to iteratively update the class prototypes to fine-tune the model until the optimal class-separating model is identified. In the following, we will analyze the principle of HDC single-pass training and the gradient descent in the iterative update process from the perspective of statistical learning and optimization. This analysis facilitates the extension of HDC into the general framework of federated learning and theoretically prove its good properties.

1.) Single-pass training: Assume each sample $\mathbf{x} \in \mathcal{X}$ has a binary label $y \in \{-1, 1\}$ for simplicity, which is easily extendable to multi-class problems via "one-versus-rest" rules. Fisher's linear discriminant in HD space finds the line $z = \mathbf{w}^T\mathbf{h}$ that maximizes class separation by optimizing the projection direction $\mathbf{w}$. The goal is for classes to be as far apart as possible, while the distance between intra-class scatter points is as small as possible with low variance. So the criterion that quantifies the desired goal can be expressed as *Rayleigh quotient*:

$$\begin{aligned} J(\mathbf{w}) &= \frac{\mathbf{w}^T \mathrm{S}_B \mathbf{w}}{\mathbf{w}^T \mathrm{S}_W \mathbf{w}} \\ \mathrm{S}_B &= (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_{-1})(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_{-1})^T \\ \mathrm{S}_W &= \Sigma_1 + \Sigma_{-1} \end{aligned} \tag{4}$$

where $\boldsymbol{\mu}_{\pm 1}$ and $\Sigma_{\pm 1}$ are the mean vector and the covariance matrix respectively. S_B is defined as the between-class scatter which measures the separation between class means, while S_W is the within-class scatter, measuring the variability inside the classes. Our goal is achieved by maximizing the Rayleigh quotient with respect to $\mathbf{w}$. The corresponding optimal projection direction is then given as: $\mathbf{w}^* = (\Sigma_1 + \Sigma_{-1})^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_{-1})$. Fisher's linear discriminant can be used as a classifier by applying a threshold to the dot product (projection) as the decision criterion:

$$z = (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_{-1})^T (\Sigma_1 + \Sigma_{-1})^{-1} \mathbf{h}_q + T \begin{cases} > 0, & \hat{y}_q = \ \ 1 \\ < 0, & \hat{y}_q = -1 \end{cases} \tag{5}$$

For two classes, the "similarity check" step in (3) can be rewritten in the form of a decision function as follows:

$$\hat{y}_q = \begin{cases} 1, & \text{if } (\frac{\mathbf{c}_1}{\|\mathbf{c}_1\|} - \frac{\mathbf{c}_{-1}}{\|\mathbf{c}_{-1}\|})^T \mathbf{h}_q > 0 \\ -1, & \text{if } (\frac{\mathbf{c}_1}{\|\mathbf{c}_1\|} - \frac{\mathbf{c}_{-1}}{\|\mathbf{c}_{-1}\|})^T \mathbf{h}_q < 0 \end{cases} \tag{6}$$

Since the class prototypes are normalized sums of hypervectors with the same labels, they relate to the respective class means by a scalar multiplication, i.e, $\mathbf{c}_{\pm 1} = \frac{\|\mathbf{c}_{\pm 1}\|}{N_{\pm 1}} \boldsymbol{\mu}_{\pm 1}$. Here, $N_{\pm 1}$ denotes the total number of samples in classes. We obtain the below decision rule after plugging in $\boldsymbol{\mu}_{\pm 1}$ into (6), then dividing both sides of the inequalities by $\frac{\|\mathbf{c}_{\pm 1}\|}{N_{\pm 1}}$.

$$\hat{y}_q = \begin{cases} 1, & \text{if } (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_{-1})^T \mathbf{h}_q > 0 \\ -1, & \text{if } (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_{-1})^T \mathbf{h}_q < 0 \end{cases} \tag{7}$$

This classifier is equivalent to Equation (5) in the special case where $\Sigma_1 = \Sigma_{-1} = \Sigma = \frac{1}{2}\mathbf{I}$. Therefore, one-shot training followed by inference in HD computing is equivalent to applying Fisher's linear discriminant and classifying sample encoded hypervectors, which involves projecting the data through high-definition coding, making it linearly separable, and then finding a linear discriminant.

2.) Iterative training: Let $\mathbf{w} \in \mathbb{R}^d$ be a vector of weights that specifies a hyperplane in the hyperdimensional space with d dimensions. We define this vector in terms of class prototypes, such that $\mathbf{w} = \mathbf{c}_1 - \mathbf{c}_{-1}$. Then, after inputing in the weight vector and simplifying the equations in (6), classification of a query data $\mathbf{x}_q$ is made through the following decision function:

$$\hat{y}_q = \begin{cases} 1, & \text{if } \mathbf{w}^T \mathbf{h}_q > 0 \\ -1, & \text{if } \mathbf{w}^T \mathbf{h}_q < 0 \end{cases} \tag{8}$$

This can be interpreted as a *linear separator* on the HD representations of the data. It divides $\mathcal{H}$ into two half-planes, where the boundary is the plane with normal $\mathbf{w}$. The goal is to learn the weights such that all the positive examples ($y_i = 1$) are on one side of the hyperplane and all negative examples ($y_i = -1$) on the other. For the optimal set of weights, the linear function $g(\mathbf{h}) = \mathbf{w}^T\mathbf{h}$ agrees in the sign with the labels on all training instances, i.e., $\text{sign}(\langle \mathbf{w}, \mathbf{h}_i \rangle) = y_i$ for any $\mathbf{x}_i \in \mathcal{X}$. We can also express this condition as $y_i\langle \mathbf{w}, \mathbf{h}_i \rangle > 0$.

Therefore, HD retraining can be represented as an instance of Empirical Risk Minimization (ERM). Particularly, we frame the retraining step as an optimization problem with convex loss function, then we argue that the updates are equivalent to stochastic gradient descent (SGD) steps over an empirical risk objective. Our ultimate goal is to find the discriminant function $g_{\mathbf{w}}(\mathbf{h})$ which minimizes the empirical risk on the embedded training set $\mathcal{D}_{\mathcal{H}} = \{(\mathbf{h}_1, y_1), ..., (\mathbf{h}_n, y_n)\}$. Empirical risk is defined as follows:

$$R_{emp}(g_{\mathbf{w}}) = \frac{1}{n}\sum_{i=1}^{n} \ell(g_{\mathbf{w}}(\mathbf{h}_i), y_i) \tag{9}$$

where $\ell : \mathcal{H} \times \mathcal{H} \longrightarrow \mathbb{R}$ is the *loss function* The "no error" condition, $y_i\langle \mathbf{w}, \mathbf{h}_i \rangle > 0\ \forall i$, provides a very concise expression for the situation of zero empirical risk. It allows for the formulation of the learning problem as the following function optimization:

$$\text{minimize}\ \ J(\mathbf{w}) = -\sum_{i=1}^{n} y_i \mathbf{w}^T \mathbf{h}_i \tag{10}$$

The solution can be obtained by applying gradient descent to minimize the cost function $J(\mathbf{w})$, with the gradient calculated as $\nabla J(\mathbf{w}) = -\sum_{i=1}^{n} y_i \mathbf{h}_i$. Alternatively, stochastic gradient descent (SGD) can be used, where a single random example is selected at each step to update the model parameters. For an individual example, the gradient is $-y_i\mathbf{h}_i$. Using a loss function $\ell(\cdot)$, the stochastic gradient descent algorithm is defined as follows: Set the starting point $\mathbf{w} = \mathbf{w}_{init}$, learning rates $\eta_1, \eta_2, \eta_3, ...$ (e.g. $\mathbf{w}_{init} = 0$ and $\eta_t = \eta$ for all t, or $\eta_t = 1/\sqrt{t}$). For a sequence of random examples $(\mathbf{h}_1, y_1), (\mathbf{h}_2, y_2), ...$ Thus, given example $(\mathbf{h}_t, y_t)$, compute the gradient $\nabla \ell(g_{\mathbf{w}}(\mathbf{h}_t), y_t)$ of the loss w.r.t. the weights $\mathbf{w}$. Update: $\mathbf{w} \longleftarrow \mathbf{w} - \eta_t \nabla \ell(g_{\mathbf{w}}(\mathbf{h}_t), y_t)$. Consider the loss function $\ell(g_{\mathbf{w}}(\mathbf{h}), y) = \max(0, y\langle \mathbf{w}, \mathbf{h} \rangle)$ for the empirical risk in (9). If $g_{\mathbf{w}}(\mathbf{h})$ has the correct sign, the loss is 0, and no update to $\mathbf{w}$ occurs since the gradient is zero. If the sign is incorrect, the gradient becomes $-y\mathbf{h}$, and the algorithm updates $\mathbf{w} \longleftarrow \mathbf{w} + \eta y \mathbf{h}$ with $\eta_t = \eta$. This is exactly the same process as HDC iterative training. Therefore, we can use the SGD with the above loss function to achieve empirical risk minimization in HDC's update process.

4 HYPERDIMENSIONAL COMPUTATION UNDER FEDERATED LEARNING FRAMEWORK

In this section, we first formalize the training algorithm for HDC under the standard federated averaging framework and then give a detailed convergence analysis and proof for the first time. Next, we implement HDC under federated learning using the PyTorch framework and test its key performance. The results of these experiments verify its good convergence, but also point out its limitations in feature extraction. Based on this analysis, we propose FHDnn, a more efficient and robust federated learning framework for HDC in Section 5.

4.1 HDC Training Algorithm under the Federated Learning Framework

Following the standard federal average framework set out in the seminal work [44], we consider one central server and a fixed set of N clients, each holding a local dataset. The k-th client, $k \in [N]$, stores embedded dataset $\mathcal{D}_k = \{(\mathbf{h}_{k,j}, y_{k,j})\}_{j=1}^{n_k}$, with $n_k = |\mathcal{D}_k|$ denoting the number of feature-label tuples in the respective datasets. The goal in federated learning is to learn a global model by leveraging the local data at the clients. The raw datasets cannot be shared with the central server due to privacy concerns, hence the training process is apportioned among the individual clients as described by the following distributed optimization problem:

$$\min_{\mathbf{w}} \left\{ F(\mathbf{w}) \triangleq \sum_{k=1}^{N} p_k F_k(\mathbf{w}) \right\} \tag{11}$$

where p_k is the weight of the k-th client such that $p_k \geq 0$ and $\sum_{k=1}^{N} p_k = 1$. A natural and common approach is to pick $p_k = \frac{n_k}{n}$. Similar to Section 3.3, we represent our HD model by a vector of parameters $\mathbf{w} \in \mathcal{H} \subseteq \mathbb{R}^d$. If the partition $\mathcal{D}_k$ is formed by randomly and uniformly distributing the training examples over the clients, then we have $\mathbb{E}_{\mathcal{D}_k}[F_k(\mathbf{w})] = F(\mathbf{w})$, where the expectation is over the set of examples assigned to the client. This is the IID assumption that usually does not hold in federated learning setting; F_k could be an arbitrarily bad approximation to F under non-IID data. We introduce a loss function as in (9) to define the learning objective and measure the fit of the model to data. Denote $\ell\big(\mathbf{w}; (\mathbf{h}_{k,j}, y_{k,j})\big)$ for the loss of the prediction on example $(\mathbf{h}_{k,j}, y_{k,j})$ made with an HD model parametrized by $\mathbf{w}$. For the k-th client, the local objective $F_k(\cdot)$ is defined in the form of local empirical loss as follows:

$$F_k(\mathbf{w}) = \frac{1}{n_k} \sum_{j=1}^{n_k} \ell\big(\mathbf{w}; (\mathbf{h}_{k,j}, y_{k,j})\big) \tag{12}$$

The local empirical loss F_k measures how well the client model fits the local data, whereas the global loss F quantifies the fit to the entire dataset on average. In Section 3, we discuss the SGD optimization of the loss function $\ell = \max(0, y\langle \mathbf{w}, \mathbf{h} \rangle)$ equivalent to the update of the HDC iterative training. Therefore, the objective is to find the model $\mathbf{w}^*$ that minimizes the global loss, i.e., $\mathbf{w}^* = \operatorname{argmin}_{\mathbf{w}} F(\mathbf{w})$. We formalize the HDC training algorithm under the federated learning framework as follows:

Algorithm. In the federated learning framework, each client maintains its own HD model and participates in building a global model that solves Equation.(11) in a distributed fashion. This is achieved via an iterative training procedure for which we describe one round (say t-th) of the algorithm below.

(1) **Broadcast:** The central server broadcasts the latest global HD model, $\mathbf{w}_t$, to all clients.
(2) **Local updates:** Each client $k \in [N]$ sets its model $\mathbf{w}_t^k = \mathbf{w}_t$ and then performs training for E epochs using local data:

$$\begin{aligned} &\mathbf{w}_{t,0}^k = \mathbf{w}_t^k, \\ &\mathbf{w}_{t,\tau+1}^k \longleftarrow \mathbf{w}_{t,\tau}^k - \eta_t \nabla F_k(\mathbf{w}_{t,\tau}^k, \xi_\tau^k), \;\; i = 0, 1, ..., E-1, \\ &\mathbf{w}_{t+1}^k = \mathbf{w}_{t,E}^k, \end{aligned} \tag{13}$$

where η_t is the learning rate and ξ_τ^k is a mini batch of data examples sampled uniformly from local dataset $\mathcal{D}_k$.
(3) **Aggregation:** The central server receives and aggregates the local models to produce a new global model:

$$\mathbf{w}_{t+1} = \sum_{k=1}^{N} p_k \mathbf{w}_{t+1}^k. \tag{14}$$

After aggregation, the server moves on to the next round, $t + 1$. This procedure is carried out until sufficient convergence is achieved. Fig. 3 summarizes the HDC training process under the federated learning framework. The overall update in one round of federated bundling is similar to a gradient descent step over the empirical loss corresponding to the entire distributed dataset across clients.

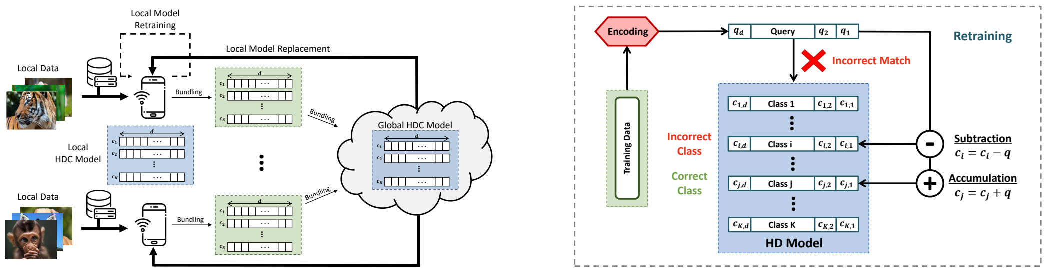


Fig. 3. HDC training process under the federated learning framework.

4.2 Convergence Analysis and Proof

For federated learning with HD algorithm, the optimization problem in (11) is cast as follows:

$$\mathbf{w}^* = \operatorname*{argmin}_{\mathbf{w}} \sum_{k=1}^{N} \frac{p_k}{n_k} \sum_{j=1}^{n_k} \max(0, y_j \langle \mathbf{w}, \mathbf{h}_j \rangle), \tag{15}$$

and the local gradient $\mathbf{g}_k = \nabla F_k(\mathbf{w})$ is computed at client $k \in [N]$ as:

$$\mathbf{g}_k = \frac{1}{n_k} \sum_{j=1}^{n_k} y_j \mathbf{h}_j \tag{16}$$

As Equation (16) suggests, the gradient computations are linear, demand low-complexity operations, and thus are favourable for resource-constrained, low-power client devices. However, in many learning tasks, linear federated learning models perform sub-optimally compared to their counterpart, DNN-based approaches.

The introduction of HDC into the federated learning framework enables the model to achieve both the superior performance of a nonlinear model and the low computational complexity of a linear model. This is a direct result of HDC, which embeds data into a high-dimensional space where the geometry is such that simple learning methods are effective. As we show in the following, linearity in HD training benefits convergence, at the same time the performance does not degrade due to the properties of non-linear hyperdimensional embeddings. Such convergence claims are not possible for non-convex and non-linear DNNs. To prove that it has a good rate of convergence, we further analyze the properties of the functions $F_k(\cdot)$ and the gradients $\nabla F_k(\cdot)$:

(1) (**L-smoothness**). Each local function $F_k(\cdot)$ is L-smooth where the gradients $\nabla F_k(\cdot)$ are Lipschitz continuous: *There exists a parameter* $L > 0$ *such that for all* $\mathbf{v},\mathbf{w} \in \mathbb{R}^d$,

$$\|\nabla F_k(\mathbf{v}) - \nabla F_k(\mathbf{w})\| \leq L\|\mathbf{v} - \mathbf{w}\|.$$

(2) (**Strong convexity**). Each local function $F_k(\cdot)$ is μ-strongly convex and differentiable: *For all* $\mathbf{v},\mathbf{w} \in \mathbb{R}^d$,

$$F_k(\mathbf{v}) \geq F_k(\mathbf{w}) + (\mathbf{v} - \mathbf{w})^T \nabla F_k(\mathbf{w}) + \frac{\mu}{2}\|\mathbf{v} - \mathbf{w}\|^2.$$

(3) (**Bounded variance**). The variance of stochastic gradients for each client k is bounded: *Let ξ^k be sampled from the k-th client's dataset uniformly at random, then there exists constants σ_k such that for all $\mathbf{w} \in \mathbb{R}^d$,*

$$\mathbb{E}\|\nabla F_k(\mathbf{w}, \xi^k) - \nabla F_k(\mathbf{w})\|^2 \leq \sigma_k^2.$$

(4) (**Uniformly bounded gradient**). The expected squared norm of stochastic gradients is uniformly bounded: *for all mini-batches ξ^k at client $k \in [N]$ and for $\mathbf{w} \in \mathbb{R}^d$,*

$$\mathbb{E}\|\nabla F_k(\mathbf{w}, \xi^k)\|^2 \leq G^2.$$

These conditions on local functions are typical and widely used for the convergence analysis of different federated averaging frameworks.

Theorem 1. *Define $\kappa = \frac{L}{\mu}$, $\gamma = max\{8\kappa, E\}$ and choose learning rate $\eta_t = \frac{2}{\mu(\gamma+t)}$. The convergence with Non-IID datasets and partial client participation satisfies:*

$$\mathbb{E}[F(\mathbf{w}_T)] - F^* \leq \frac{2\kappa}{\gamma + T}\left[\frac{B}{\mu} + \left(2L + \frac{E\mu}{4}\right)\|\mathbf{w}_0 - \mathbf{w}^*\|^2\right] \tag{17}$$

where

$$B = \sum_{k=1}^{N} p_k^2 \sigma_k^2 + 6L\Gamma + 8(E-1)^2 G^2 + \frac{N-K}{N-1}\frac{4}{K}E^2 H^2 \tag{18}$$

Here, T is the number of communication rounds (or SGD steps), The term Γ is used to quantify the degree of Non-IID [42]. Let F^* and F_k^* be the minimum values of F and F_k, respectively, then $\Gamma = F^* - \sum_{k=1}^{N} p_k F_k^*$. As shown in Theorem 1, HDC under federated learning framework can achieve $\mathcal{O}(\frac{1}{T})$ convergence rate. Detailed derivation and proof procedures are shown in Appendix A.

4.3 The Performance Test Experiment

We study a pre-experiment to test the actual performance of the HDC training algorithm in a generalized federated learning scenario. Specifically, we implemented HDC in a federated learning framework using the PyTorch framework and compared it to a lightweight fully connected layer neural network. We used hypervectors of dimension 10,000 to encode the data. The fully connected layer neural network has 128 neurons and ReLU activation units, and a final output layer with softmax. To observe the performance of them focusing on the real-world use-cases, we evaluated it on a wide range of benchmarks shown in Table 1 that range from relatively small datasets collected in a small IoT network to a large dataset that includes hundreds of thousands of face images. The dataset tested is shown in Table 1. **ISOLET:** recognizing audio of the English alphabet, **UCIHAR:** detecting human activity based on 3-axial linear acceleration and angular velocity data, from different people, **PAMAP2:** classifying five human activities based on a heart rate and inertial measurements, **FACE:** classifying images with faces/non-faces, and **MNIST:** recognizing handwritten digits by different people.

Table 1. Datasets (n: Feature Size, K: Number of Classes)

Dataset	n	K	**Train Size**	**Test Size**	**Description**
ISOLET	617	26	6,238	1,559	Voice Recognition
UCIHAR	561	12	6,213	1,554	Activity Recognition (Mobile)
PAMAP2	75	5	611,142	101,582	Activity Recognition (IMU)
FACE	608	2	522,441	2,494	Face Recognition
MNIST	784	10	60,000	10,000	Handwritten Digit Recognition

4.3.1 Accuracy and Convergence Results. We conduct our experiments for 100 clients and 100 rounds of communication. We first tune the hyperparameters for both HDC and CNNs, then experiment with different federated learning parameters. Fig. 4 shows the accuracy and convergence for different values of local epochs E and local batch sizes B. For all experiments, $C = 0.2$ fraction of clients are randomly picked in every communication round. For all datasets, the best convergence is achieved with a small number of epochs ($E = 1$) and moderate batch sizes ($B = 10, 20$). As shown in Fig. 4, HDC and NN achieve high accuracy on other three datasets except the PAMAP2 dataset. Compared with NN, HDC shows a faster convergence rate and can provide faster early results. NN, on the other hand, can benefit more from larger local training periods and achieve better optimal accuracy. For the more challenging PAMAP2 dataset, the accuracy of both HDC and NN decreased. The HDC converges quickly but fluctuates greatly, and neural networks consistently outperform HDC. It shows the difficulty of HDC in extracting effective features when performing more complex analyses, which severely limits its accuracy.

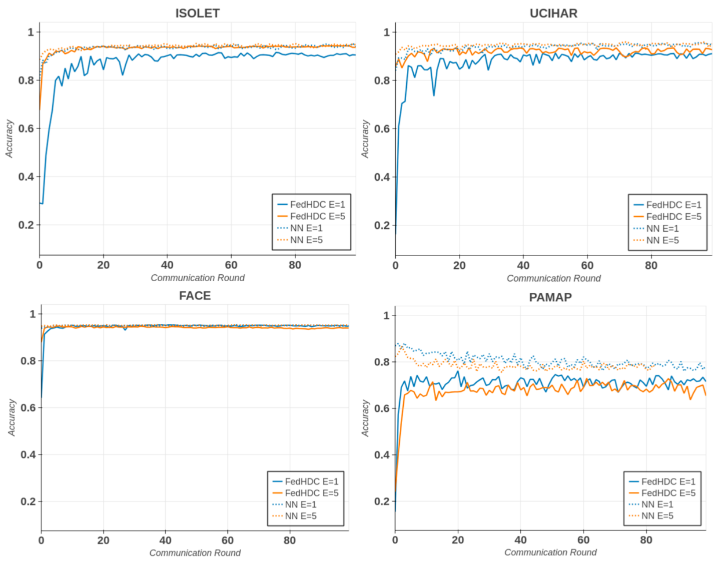


Fig. 4. Accuracy and convergence for various epochs (E).

4.3.2 Impact of Hypervector Dimensionality on Accuracy. Understanding the relationship between the dimension of high-dimensional vector and problem complexity is crucial for optimizing the performance of high-dimensional computational methods in various classification tasks. We test the effect of hypervector dimensions on HDC classification accuracy in federated learning framework,as shown in Table 2. It can be observed that the accuracy increases, and as the number of dimensions increases. Some previously work [54] showed that dimensionality is directly proportional to the bandwidth of the noise in HDC classification problems, thus providing a guideline for a trade off between noise and the hypervector size. It is essential to consider the trade-off between performance and resource usage, as the computational cost rises with increasing dimensions.

In essence, the HD encoding dimension exhibits a linear relationship with the number of categorical features, while it depends logarithmically on the alphabet size. The separation quality of the problem is associated with factors such as the class separability and the encoding dimension. Intuitively, when the classes are well separated, a smaller encoding dimension can be employed to achieve satisfactory performance. This is because the inherent separability of the data aids in reducing the required dimensionality for efficient classification. Conversely, when the classes are poorly separated, a larger encoding dimension is necessary to enhance the robustness and accuracy of the classification process.

Table 2. Impact of Dimensionality on Accuracy

Dataset	**1000**	**2000**	**4000**	**8000**	**10000**
ISOLET	90.79%	93.36%	95.07%	95.37%	94.59%
UCIHAR	90.60%	93.98%	93.63%	93.54%	94.46%
PAMAP2	74.9%	76.88%	76.10%	77.85%	77.98%
FACE	95.05%	95.2%	95.74%	95.86%	96.17%
MNIST	92.24%	93.81%	95.37%	96.34%	96.80%

5 FHDNN: A SYNERGETIC FEDERATED HYPERDIMENSIONAL COMPUTING BASED ON CNN AND HDC

From the experimental results in Section 4.3, the results indicate that although having faster convergence, is not as accurate as NN, especially on challenging datasets. This is even more evident when performing complex image analysis, for example, Table 3 summarizes the accuracy of various current HDC coding methods when running the image classification task [12]. The results show that current HDC coding methods are unable to match state-of-the-art accuracy, mainly due to the difficulty of extracting accurate features by HDC. In this section, to overcome this problem, we propose FHDnn, a synergistic federated learning framework that combines CNN and HDC. FHDnn uses a pre-trained CNN as a feature extractor, whose outputs are encoded into hypervectors, which are then used for training. CNNs excel at learning complex hierarchies of features with high accuracy, but in the real application scenario, where transmission costs and resources are limited, performing federated learning is prone to errors. On the other hand, HDC can provide efficient and robust training for such distributed computation with large-scale communication. FHDnn effectively combines their complementary strengths to realize a lightweight, communication-efficient and highly robust federated learning framework.

Table 3. Accuracy of HDC on image datasets

Model	CIFAR10	CIFAR100	Flowers	dtd	GTSRB
HD-linear	26.94	8.98	19.58	6.41	83.63
HD-non linear	41.98	20.35	25.68	8.13	84.11
HD-id level	26.56	9.45	15.97	6.25	44.86
CNN	90.1	78.4	81	98.7	94.6

5.1 Model Architecture

FHDnn uses a horizontal federated learning architecture. Each client trains the model locally based on the data it has, which has the same features. Subsequently, the clients upload the trained model parameters to a central server, which aggregates the model parameters from all clients to form a unified global model. FHDnn consists of two components: *1).* a pre-trained CNN as a feature extractor and *2).* a federated HD learner. Fig. 5 shows the model architecture of FHDnn. The pre-trained feature extractor is trained once and not updated at run time. This removes the need for costly CNN weight updates via federated learning. Instead, HD Computing is responsible for all the federated model updates. Thus, it is much more efficient and scalable. Next, we describe these two components in detail.

Feature Extractor: Any standard CNN model can be used as a feature extractor, in this work we have chosen the pre-trained SimCLR ResNet model due to its excellent performance. SimCLR [7] is a contrastive learning framework which learns representations of images in a self-supervised manner by maximizing the similarity between latent space representations of different augmentations of a single image. This class-agnostic framework trained on a large image dataset allows for transfer learning over multiple datasets, (as evaluated in [7]) making it ideal for a generic feature extractor. Standard CNNs learn representations that are fine-tuned to optimize the classification performance of the dense classifier at the end of the network. Since SimCLR focuses on learning general representations as opposed to classification oriented representations, it is a good choice of a feature extractor. It is also possible to use other models such as MobileNet [20].

HD Learner: FHDnn encodes the outputs of the feature extractor into hypervectors. More formally, given a point $\mathbf{x} \in \mathcal{X}$, the features $\mathbf{z} \subset \mathbb{Z}^n$ are extracted using the feature extractor $f : \mathcal{X} \to \mathbb{Z}$ where f is a pre-trained neural network. The HD embedding is constructed as $\mathbf{h} = \phi(\mathbf{z}) = \text{sign}(\mathbf{\Phi z})$ under the encoding function $\phi : \mathbb{Z} \to \mathcal{H}$. HD learner then operates on these hypervectors using binding and bundling which are simple and highly parallelizable. The goal of such configuration is to avoid the transmission of the CNN and instead train only the HD learner in a federated manner. An HD model is formed by bundling all encoded hypervectors with the same class level together. We perform bundling by the element-wise addition of those hypervectors, which generates corresponding class protototypes. Thus, the HD model is simply a set of hypervectors with the number of classes in the dataset. Then, we use the HD learner in federated training.

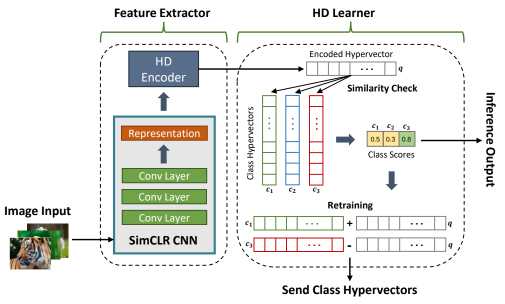


Fig. 5. FHDnn Model Architecture

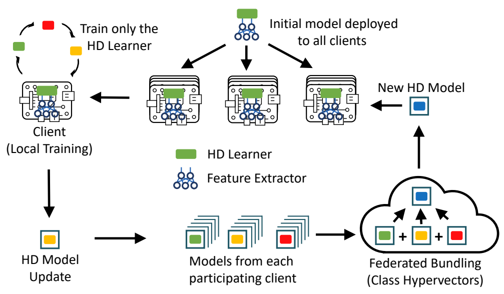

Train only the HD Learner
Initial model deployed to all clients
Client (Local Training)
New HD Model
HD Learner
Feature Extractor
HD Model Update
Models from each participating client
Federated Bundling (Class Hypervectors)

Fig. 6. FHDnn Federated Training

5.2 Federated Training

Fig. 6 summarizes the overall federated training process for FHDnn. The whole process involves two steps, client local training and federated bundling. These two steps work in a cyclical fashion, one after the other, until convergence.

Client Local Training: Each client initially starts the process with a feature extractor f and an untrained HD learner. Once we get the encoded hypervectors, we create class prototypes by bundling together hypervectors of the corresponding class using $\mathbf{c}_k = \sum_i \mathbf{h}_i^k$. Inference is done by computing the cosine similarity metric between a given encoded data point with each of the prototypes, returning the class which has maximum similarity. After this one-shot learning process, we iteratively refine the class prototypes by subtracting the hypervectors from the mispredicted class prototype and adding it to the correct prototype as shown in Fig. 5. We define the complete HD model $\mathbf{C}$ as the concatenation of class hypervectors, i.e., $\mathbf{C} = [\mathbf{c}_1^T, \mathbf{c}_2^T, ..., \mathbf{c}_l^T]$.

Federated Bundling: In the federated bundling framework, each client maintains its own HD model and participates to build a global model in a distributed fashion. This is achieved via an iterative training procedure for which we describe one round (say t-th) of the algorithm below.

(1) *Broadcast:* The central server broadcasts the latest global HD model, $\mathbf{C}^t$, to all clients.
(2) *Local updates:* Each participating client $k \in [N]$ sets its model $\mathbf{C}_t^k = \mathbf{C}_t$ and then performs training for E epochs using local data.
(3) *Aggregation:* The central server receives and aggregates the local models to produce a new global model:

$$\mathbf{C}_{t+1} = \sum_{k=1}^{N} \mathbf{C}_{t+1}^k. \tag{19}$$

After aggregation, the server moves on to the next round, $t + 1$. This procedure is carried out until sufficient convergence is achieved.

5.3 Federated learning Over Unreliable Channels With FHDnn

Federated learning often occurs over unreliable wireless channels, introducing signal noise and errors. We study how FHDnn provide reliable learning over such networks without additional overhead. Following a common assumption in recent work [5, 29, 42, 49, 53], we assume that the server makes error-free downstream broadcasts. For uplink, we consider a constrained multiple access channel (MAC) shared among clients. To manage interference, we employ orthogonal frequency division multiple access (OFDMA) [17], allocating dedicated resource blocks to each client. While this ensures separate model recovery, individual channels remain noisy. Conventional approaches limit rates for error-free transmission under Shannon's theorem, but this scales poorly with increasing client numbers, reducing throughput and training speed. Instead, we allow errors in server-received models, leveraging the learning algorithm's tolerance for perturbations. We analyze FHDnn's performance over unreliable MAC with corrupted model transmissions, considering three error models at different network stack layers. We then explore FHDnn properties that how to enhance learning robustness under these conditions and design techniques for further improvement.

5.3.1 Noisy Aggregation. In conventional procession, the transmitter performs three steps to generate the wireless signal from data: source coding, channel coding, and modulation. First, a source encoder removes the redundancies and compresses the data. Then, to protect the compressed bitstream against the impairments introduced by the channel, a channel code is applied. The coded bitstream is finally modulated with a modulation scheme which maps the bits to complex-valued samples (symbols), transmitted over the communication link.

The receiver inverts the above operations, but in the reverse order. A demodulator first maps the received complex-valued channel output to a sequence of bits. This bitstream is then decoded with a channel decoder to obtain the original compressed data; however, it might be possibly corrupted due to the channel impairments. Lastly, the source decoder provides a (usually inexact) reconstruction of the transmitted data by applying a decompression algorithm.

For noisy aggregation, as an alternative of the conventional pipeline, we assume uncoded transmission [15]. This scheme bypasses the transformation of the model to a sequence of bits, which are then need to be mapped again to complex-valued channel inputs. Instead, the real model parameter values are directly mapped to the complex-valued samples transmitted over the channel. Leveraging the properties of uncoded transmission, we can treat the channel as formulated in Equation (20), where the additive noise is directly applied to model parameters. The channel output received by the server for client k at round t is given by:

$$\tilde{\mathbf{w}}_t^k = \mathbf{w}_t^k + \mathbf{n}_t^k \tag{20}$$

where $\mathbf{n}_t^k \sim \mathcal{N}(0,\, \sigma_{t,k}^2)$ is the d-dimensional additive noise. The signal power and noise power are computed as $\mathbb{E}\|\mathbf{w}_t^k\|^2 = P_{t,k}$ and $\mathbb{E}\|\mathbf{n}_t^k\|^2 = \sigma_{t,k}^2$, respectively. The signal-to-noise ratio (SNR) is:

$$SNR_{t,k} = \frac{\mathbb{E}\|\mathbf{w}_t^k\|^2}{\mathbb{E}\|\mathbf{n}_t^k\|^2} = \frac{P_{t,k}}{\sigma_{t,k}^2} \tag{21}$$

An immediate result of federated bundling is the improvement in the SNR for the global model. When the class hypervectors from different clients are bundled at the server, the signal power scales up quadratically with the number of clients N, whereas the noise power scales linearly. Assuming that the noise for each client is independent, we have the following relation:

$$SNR_t = \frac{\mathbb{E}\big[\sum_{k=1}^N \mathbf{w}_t^k\big]}{\mathbb{E}\big[\sum_{k=1}^N \mathbf{n}_t^k\big]} \approx \frac{N^2 P_{t,k}}{N\sigma_{t,k}^2} = N \times SNR_{t,k} \tag{22}$$

Notice that the effect of noise is suppressed by N times due to bundling. This claim can also be made for the FedAvg [33] framework over CNNs. However, even though the noise reduction factor is the same, the impact of the small noise might be amplified by large activations of CNN layers. In FHDnn, we do not have such problem as the inference and training operations are purely linear.

One other difference of FHDnn from CNNs is its information dispersal property. HD encoding produces hypervectors which have holographic representations, meaning that the information content is spread over all the dimensions of the high-dimensional space and no dimension in a hypervector is more responsible for storing any piece of information than others. Since the noise in each dimension can be also assumed independent, we can leverage the information spread to further eliminate noise.

Consider the random projection encoding described in Section 3.1, which is also illustrated by Fig. 2a. Let the encoding matrix $\mathbf{\Phi} \in \mathbb{R}^{d\times n}$ expressed in terms of its d row vectors, i.e., $\mathbf{\Phi} = [\Phi_1, \Phi_2, ..., \Phi_d]^T$. Then, the hypervector formed by encoding information $\mathbf{x} \in \mathcal{X}$ can be written as $\mathbf{h} = [\Phi_1^T\mathbf{x}, \Phi_2^T\mathbf{x}, ..., \Phi_d^T\mathbf{x}]^T$, where $\mathbf{x} = [x_1, x_2, ..., x_n]^T$. As implied by this expression, the information is dispersed over the hypervectors uniformly. Now consider additive noise over the same hypervector such that $\mathbf{h} + \mathbf{n} = [\Phi_1^T\mathbf{x} + n_1, \Phi_2^T\mathbf{x} + n_2, ..., \Phi_d^T\mathbf{x} + n_d]^T$. We can reconstruct the encoded information from the noisy hypervector $\tilde{\mathbf{h}} = \mathbf{h} + \mathbf{n}$ as follows:

$$\mathbf{x} \approx \Big[\frac{1}{d}\sum_{i=1}^{d}\Phi_{i,1}\tilde{\mathbf{h}}_i, \frac{1}{d}\sum_{i=1}^{d}\Phi_{i,2}\tilde{\mathbf{h}}_i, ..., \frac{1}{d}\sum_{i=1}^{d}\Phi_{i,n}\tilde{\mathbf{h}}_i\Big] \tag{23}$$

where $\tilde{\mathbf{h}}_i = \Phi_i^T\mathbf{x} + n_i$ are the elements of the noisy hypervector. The noise variance is then reduced by the averaging operation, similar to the case in Equation (22). Therefore, in HD computing, the noise is not only suppressed by bundling accross models from different clients, but also by averaging over the dimensions within the same hypervector. We demonstrate this over an example where we encode a sample from the MNIST dataset, add Gaussian noise, then reconstruct it. Fig. 7 shows the original image, noisy image in the sample space, and reconstructed image for which the noise was added in the hyperdimensional space.

Finally, there is a “flying under the radar” principle for federated learning over noisy channel. The analysis in [58] points out since SGD is inherently a noisy process, as long as the channel noise do not dominate the SGD noise during model training, the convergence behavior is not affected. As a result, FHDnn’s can maintain good convergence performance since the noise is greatly suppressed during the training process.

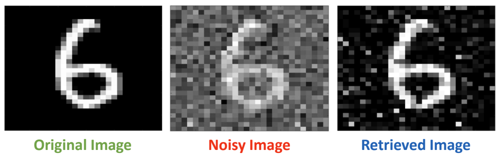


Fig. 7. Noise robustness comparison

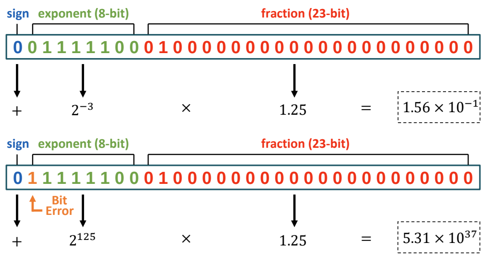


Fig. 8. Single bit error on a floating-point number

5.3.2 Bit Errors. In a federated learning scenario, since the central server needs to aggregate parameters from each client and continuously update the global model, bit inconsistencies during communication with any client may cause the final model to fail. Therefore, Bit Error Rate (BER) is an important metric to evaluate the robustness of federated learning systems.

BER quantifies how accurately the receiver can decode transmitted data by measuring the frequency of bit errors, which are instances where the received digital symbols differ from their transmitted counterparts. These errors are typically assessed by the Hamming distance between the input bitstream of the channel encoder and the output bitstream of the channel decoder. Let $\hat{\mathbf{w}}$ represent the binary-coded model parameters communicated to the server. To model bit errors, the communication channel is assumed to be a binary symmetric channel (BSC), which flips each bit in $\hat{\mathbf{w}}$ independently with a probability p_e (e.g., $0 \longrightarrow 1$). The received bitstream at the server for client k during round t is then given as:

$$\tilde{\hat{\mathbf{w}}}_t^k = \hat{\mathbf{w}}_t^k \oplus \mathbf{e}_t^k \tag{24}$$

where $\mathbf{e}_t^k$ is the binary error vector and $\oplus$ denotes modulo 2 addition. Given a specific vector $\mathbf{v}$ of Hamming weight $\mathrm{wt}(\mathbf{v})$, the probability that $\mathbf{e}_t^k = \mathbf{v}$ is given by

$$\mathbb{P}(\mathbf{e}_t^k = \mathbf{v}) = p_e^{\mathrm{wt}(\mathbf{v})}(1 - p_e)^{m-\mathrm{wt}(\mathbf{v})} \tag{25}$$

The bit error probability, p_e, is a function of both the modulation scheme and the channel coding technique. Assuming lossless source coding, to conclude the transmission, the corrupted bit stream in Equation (24) is ultimately reconstructed to a real-valued model, i.e., $\tilde{\hat{\mathbf{w}}}_t^k \longrightarrow \tilde{\mathbf{w}}_t^k$.

Bit errors can have a detrimental effect on the training accuracy, especially for CNNs. At worst case, a single bit error in one client in one round can fail the whole training. In Fig. 8 we give an example of how much difference a single bit error can make for the standard 32 bit floating point CNN weights. In floating point notation, a number consists of three parts: a sign bit, an exponent, and a fractional value. In *IEEE 754* floating point representation, the sign bit is the most significant bit, bits 31 to 24 hold the exponent value, and the remaining bits contain the fractional value. The exponent bits represent a power of two ranging from -127 to 128. The fractional bits store a value between 1 and 2, which is multiplied by 2^{exp} to give the decimal value. This example shows that one bit error in the exponent can change the weight value from 0.15625 to 5.31×10^{37}.

The bit errors are contagious because a parameter from one client gets aggregated to the global model, then communicated back to all clients. Furthermore, errors propagate through all communication rounds because local training or aggregation does not completely change the parameter value, but only apply small decrements. For instance, assume a federated learning scenario with 100 clients and one bit error in a client's model as in the above example. After 10 rounds of training, the CNN weight for the global model will be on the order of $\sim \frac{5.31\times10^{37}}{100^{10}} = 5.31 \times 10^{17}$, still completely failing the whole model. Consider ResNet-50, which has 20 million parameters, so training 100

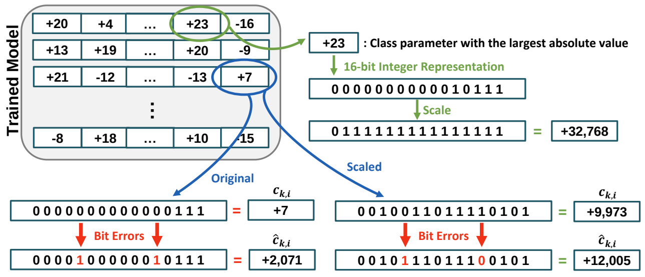


Fig. 9. An example scaling up operation

clients even over a channel with $p_e = 10^{-9}$ BER results in two errors per round on average, making model failure inevitable.

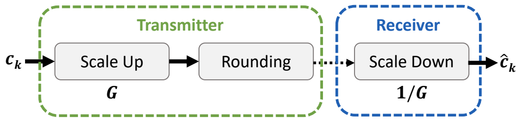


Fig. 10. Quantizer scheme

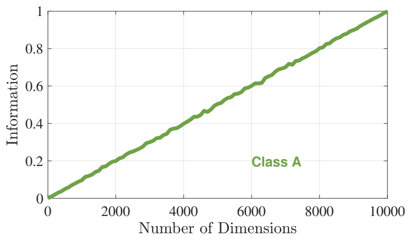
A similar problem exists with HD model parameters, but to a lesser extent because the hypervector encodings use integer representations. Fig. 9 implies that the parameters can also change significantly for the HD model. Particularly, errors in the most significant bits (MSB) of integer representation lead to higher accuracy drop. We propose a quantizer solution to prevent this issue. The adopted quantizer design is illustrated in Fig. 10. Inspired by the classical quantization methods in communication systems, we leverage *scaling up* and *scaling down* operations at the transmitter and the receiver respectively. This can be implemented by the automatic gain control (AGC) module in the wireless circuits. For a class hypervector $\mathbf{c}_k$, $k \in \{1, \ldots, K\}$, the quantizer output $Q(\mathbf{c}_k)$ can be obtained via the following steps:

(1) **Scale Up:** Each dimension in the class hypervector, i.e. $c_{k,i}$, is amplified with a scaling factor denoted quantization gain G. We adjust the gain such that the dimension with the largest absolute value attains the maximum value attainable by the integer representation. Thus, $G = \frac{2^{B-1}-1}{\max(c_k)}$, where B is the bitwidth.
(2) **Rounding:** The scaled up values are truncated to only retain their integer part.
(3) **Scale Down:** The receiver output is obtained by scaling down with the same factor G.

In this way, bit errors are applied to the scaled up values. Intuitively, we limit the impact of the bit error on the models. The prediction is realized by a normalized dot-product between the encoded query and class hypervectors, according Equation (3). Therefore, the ratio between the original parameter and the received (corrupted) parameter determines the impact of the error on the dot-product. Without our quantizer, this ratio can be very large whereas after scaling *up* then later *down*, it is diminished. Fig. 9 demonstrates this phenomenon. The ratio between the corrupted and the original parameter is $\frac{\hat{c}_{k,i}}{c_{k,i}} = \frac{2{,}071}{7} \approx 295.9$. The ratio decreases to only $\frac{\hat{c}_{k,i}}{c_{k,i}} = \frac{12{,}005}{9{,}973} \approx 1.2$ between the scaled versions.

5.3.3 Packet Loss. At the physical layer of the network stack, errors manifest as additive noise or bit flips directly affecting the transmitted data. In contrast, at the network and transport layers, errors are typically observed as packet losses. The interplay of network and protocol specifications defines the overall error characteristics, which the data transmission process must accommodate and mitigate. The type of errors allowed, whether bit errors or packet losses, is determined by the error control mechanism. For the previous error model, we assumed that the bit errors are permitted to propagate through the transport hierarchy. This assumption holds for a class of protocols designed for error-resilient applications capable of handling bit errors [56]. In some protocols, the reaction of the system to any number of bit errors is to drop the corrupted packets [37]. These protocols typically employ error-detection mechanisms such as cyclic redundancy checks (CRC) or checksums to identify bit errors. In such cases, the communication channel can be considered bit-error-free but packet-lossy. To evaluate performance in this context, we use the packet error rate (PER) as a key metric, with its expected value referred to as the packet error probability, p_p. For a packet of length N_p bits, this probability can be expressed as: $p_p = 1 - (1 - p_e)^{N_p}$.

The common solution for dealing with packet losses and guarantee successful delivery is to use a reliable transport layer communication protocol, e.g., transmission control protocol (TCP), where various mechanisms including acknowledgment messages, retransmissions, and time-outs are employed. To detect and recover from transmission failures, these mechanisms incur considerable communication overhead. Therefore, for our setup we adopt user datagram protocol (UDP), another widely used transport layer protocol. UDP is low-latency and have much less overhead compared to TCP, but it is unreliable and cannot guarantee packet delivery. HDC's information dispersal and holographic representation properties are also beneficial for packet losses. Another direct result of these concepts is obtaining partial information on data from any part of the encoded information. The intuition is that any portion of holographic coded information represents a blurred image of the entire data. Then, each transmitted symbol–packets in our case–contains an encoded image of the entire model.



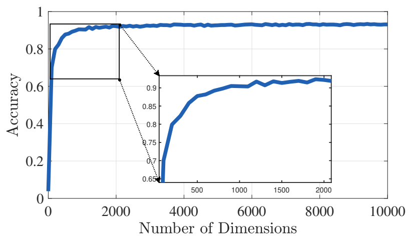


Fig. 11. Impact of partial information on similarity check (left) and classification accuracy (right)

We demonstrate the property of obtaining partial information as an example using a speech recognition dataset [1]. As Fig. 11(a) showing, after training the model, we remove the dimensions of a certain class hypervector in a random fashion. Then we perform a similarity check to figure out what portion of the original dot-product value is retrieved. The same figure shows that the amount of information retained scales linearly with number of remaining dimensions. Fig. 11(b) further clarifies our observation. We compare the dot-product values across all classes and find the class hypervector with the highest similarity. Only the relative dot-product values are important for classification. So, it is enough to have the highest dot-product value for the correct class, which holds true with $\sim$ 90% accuracy even when 80% of the hypervector dimensions are removed.

5.4 Strategies for Improving Communication Efficiency

Although the HDC model is much smaller than the DNN model, it still puts some pressure on IoT devices with limited resources in large-scale distributed communication. In order to further improve the communication efficiency of FHDnn, we further investigate the structure and properties of the class of hypervectors and design three methods: *1).* binarized differential transmission, *2).* subsampling, and *3).* sparsification & compression.

5.4.1 Binarized Differential Transmission. At the beginning of each round, the central server broadcasts the latest global HD model, $\mathbf{C}_t$, to all clients. Then, before performing local updates, each client makes a copy of this global model. Instead of sending the local updated models $\mathbf{C}^k_{t+1}$ at the aggregation step, the clients send the difference between the previous model and the updated model, i.e., $\mathbf{C}^k_{t+1} - \mathbf{C}_t$. We call this operation *differential transmission.* As shown in Equation (26), we binarize the difference to reduce the communication cost by 32x, going from 32-bit floating point to 1-bit binary transmission.

$$\Delta\mathbf{C}^k_{bin} = \text{sign}(\mathbf{C}^k_{t+1} - \mathbf{C}_t),\ \forall k \tag{26}$$

The central server receives and aggregates the differences, then adds it to the global model as:

$$\mathbf{C}_{t+1} = \mathbf{C}_t + \sum_{k=1}^{N} \mathbf{C}^k_{bin} \tag{27}$$

This global model is broadcasted back to the clients. Such binarization framework is not possible for the original federated bundling approach where clients communicate their full models. Binarizing the models itself instead of the 'difference' results in unstable behavior in training. Therefore, we utilize binarized differential transmission whose stability can be backed by studies on similar techniques. Some previous work [4] has shown that in distributed optimization, only passing the symbols of each small batch of random gradients can achieve the convergence rate of full-precision SGD-level.

5.4.2 Subsampling. In this approach, the clients only send a subsample of their local model to the central server. Each client forms and communicates a subsample matrix $\hat{\mathbf{C}}^k_{t+1}$, which is formed from a random subset of the values of $\mathbf{C}^k_{t+1}$. The server then receives and averages the subsampled client models, producing the global update $\mathbf{C}_{t+1}$ as: $\hat{\mathbf{C}}_{t+1} = \frac{1}{N}\sum_{k=1}^{N} \hat{\mathbf{C}}^k_{t+1}$. The subsample selection is completely randomized and independent for each client in each round. Therefore, the average of the sampled models at the server is an unbiased estimator of their true average, i.e., $\mathbb{E}\|\hat{\mathbf{C}}_t\| = \mathbf{C}_t$. We can achieve the desired improvement in communication by changing the subsampling rate. For example, if we subsample 10% of the values of $\mathbf{C}^k_{t+1}$, the communication cost is reduced by 10x.

5.4.3 Sparsification & Compression. The objective of this approach is to identify and remove the elements (dimensions) of each class hypervector that have minimal impact on model performance. As discussed in Section 3.2, given a query hypervector, inference is done by comparing it with all class hypervectors to find the one with the highest similarity. The similarity is typically taken to be the cosine similarity and calculated as a normalized dot-product between the query hypervector and class hypervectors. The elements of a query hypervector are input dependent and changes from one input to another one. Due to the randomness introduced by HDC encoding, the query hypervectors, on average, have a uniform distribution of values in all dimensions. Under this assumption, we need to find and drop the elements of class hypervectors that have minimal impact on cosine similarity. To identify the least impactful elements of each class hypervector, we select those with the smallest absolute values and set them to zero since they have the least contribution to the dot-product computation of cosine similarity. For instance, in the case of the i^{th} class hypervector, we identify S elements with the minimum absolute values as follows: $\min\{c^i_d, ..., c^i_2, c^i_1\}_S$. To make a model with $S\%$ sparsity, we make $\frac{S}{100} \times d$ elements of each class hypervector zero. Then, we employ the

Compressed Sparse Column (CSC) [43] to compress the sparse model. CSC stores only the non-zero data values and the number of zero elements between two consecutive non-zero elements.

6 EXPERIMENTAL ANALYSIS

In this section, we demonstrate the good performance of FHDnn through systematic experiments. We first present the dataset and settings used for evaluation and test the performance of FHDnn in a reliable communication scenario (Section 6.1). We focus on the comparison of FHDnn and CNN in accuracy, communication efficiency and energy consumption (Section 6.2-6.4). Next, we test and analyze the performance of FHDnn under three different unreliable network settings: packet loss, noise injection, and bit errors (Section 6.5). Finally, we further evaluate the 3 proposed strategies for improving communication efficiency in Section 6.6, including binarized differential transmission, subsampling, and sparsification & compression.

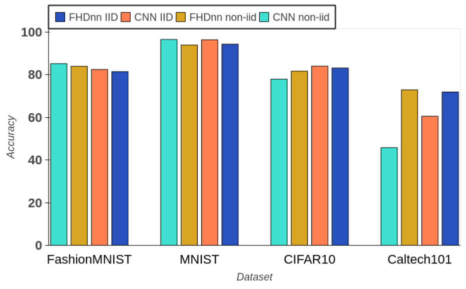


Fig. 12. Accuracy of FHDnn and ResNet on different datasets

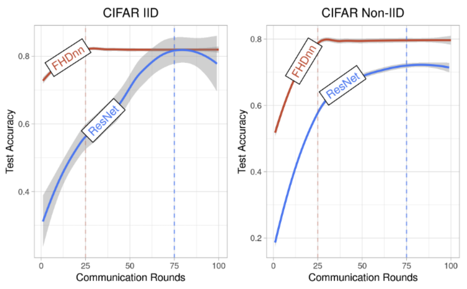


Fig. 13. Accuracy and Number of communication rounds for various hyperparameters

6.1 Experimental Setup

We evaluate FHDnn on MNIST [11], FashionMNIST[60], CIFAR10 [36], and Caltech101 [39] datasets using PyTorch. As introduced in Section 5.1, FHDnn utilizes a pre-trained SimCLR model to extract initial feature vectors, which are subsequently mapped to high-dimensional encodings. Then, the HD learner employs these high-dimensional encodings to train and update the entire federated learning model. The HD learner model is randomly initialized. To compare the performance of FHDnn and the original CNN, we compare two classical CNN networks. Specifically, we used ResNet-18 [18] to compare on FashionMNIST, CIFAR10 and Caltech101 datasets. For MNIST, since this dataset is too small, deep networks like ResNet are prone to overfitting, we use a simpler network (CNN with two 5x5 convolutional layers, two fully connected layers, and ReLU activation). Our simulations involve an IoT network with $N = 100$ clients and one server, running for 100 communication rounds. We tune hyperparameters E (local training epochs), B (local batch size), and C (fraction of participating clients), applying the best ResNet parameters to FHDnn for direct comparison. Data partitioning is examined under both IID (evenly shuffled) and Non-IID (label-sorted shards) conditions. Performance evaluations are conducted on Raspberry Pi 4[46] (quad-core Cortex-A72, 1.5GHz, 4GB RAM) and NVIDIA Jetson[9] (quad-core Cortex-A57 CPU, 128-core Maxwell GPU, 4GB RAM) to assess FHDnn's efficiency on edge devices. We repeated the experiment 100 times with different random seeds and report the average results to avoid the influence of randomness on the results.

6.2 Overall Accuracy of FHDnn

Fig. 12 compares the test accuracy of FHDnn with ResNet on MNIST, CIFAR-10, FashionMNIST, and Caltech101 datasets after 100 federated training rounds. For FashionMNIST and MNIST, the accuracy of FHDnn is comparable to that of ResNet in both IID and non-IID scenarios. The accuracy consistently exceeds 80%, demonstrating the good performance of FHDnn in handling complex image datasets. On CIFAR-10 and Caltech101, the accuracy of FHDnn is higher or comparable to ResNet for both IID and non-IID settings. Especially for non-iid data, on the Caltech101 dataset, ResNet cannot maintain accuracy, while FHDnn performs much better. Fig. 13 shows test accuracy over communication rounds for CIFAR-10, displaying the smoothed conditional mean across hyperparameters (E,B,C) for IID and Non-IID distributions. Experiments demonstrate that FHDnn not only achieves superior accuracy, but also greatly reduces the number of required communication rounds, allowing it to quickly achieve convergence even in challenging non-iid settings. For instance, FHDnn reaches near-optimal accuracy in fewer than 25 communication rounds, maintaining a steady performance thereafter. ResNet, however, requires significantly more communication rounds (≈ 75) to achieve its stable level of accuracy. As a result, FHDnn significantly reduces the reliance on extensive communication with the same target accuracy, which a key advantage in distributed systems such as in an FL scenario.

6.3 Energy Consumption of FHDnn

Local training is computationally expensive for constrained IoT devices, which was one of the main drivers for centralized learning over many years. Particularly, CNN training involves complicated architectures and backpropagation operation that is very compute intensive. In addition, this has to be repeated for many communication rounds. HD on the contrary is lightweight, low-power, and fast. Table 4 quantitatively compares the computation time and energy consumption of FHDnn and ResNet local training on 2 different edge device platforms. On Raspberry Pi, FHDnn achieves a 35% reduction in training time and energy consumption compared to ResNet, while on Nvidia Jetson, it outperforms ResNet with an 80% reduction in both metrics. These results underscore that FHDnn has significant advantages over ResNet, enabling faster and more sustainable edge-based training. It's lightweight and energy-efficient design, making it highly suitable for IoT scenarios where computational and power resources are limited.

Table 4. Performance of FHDnn on Edge Devices

Device	**Training Time (Sec)**		**Energy (J)**	
	FHDnn	ResNet	FHDnn	ResNet
Raspberry Pi	**858.72**	1328.04	**4418.4**	6742.8
Nvidia Jetson	**15.96**	90.55	**96.17**	497.572

6.4 Communication Efficiency of FHDnn

We compare the communication efficiency of FHDnn with ResNet in federated learning, targeting an accuracy of 80%. The data transmitted per client is calculated as $data_{transmitted} = n_{rounds} \times update_{size}$. The number of parameters of ResNet-18 model is about 11.7MB. With 75 rounds of communication, the client needs to send about 3.51GB of data. In comparison, the update size of FHDnn is only 1MB, and the convergence speed is 3 times faster than that of ResNet-18. Therefore, the total communication cost for the client is only about 25MB. FHDnn also significantly reduces training time. Under LTE networks with 5dB SNR, 5MHz bandwidth, and 10ms frame duration, FHDnn converges in 1.1 hours for CIFAR IID and 3.3 hours for CIFAR Non-IID. In contrast, ResNet takes 374.3 hours for both scenarios. This improvement is due to FHDnn's ability to communicate at a rate of 5.0 Mbits/sec by tolerating errors, whereas ResNet achieves only 1.6 Mbits/sec under error-free communication.

6.5 Performance of FHDnn in Unreliable Communication

In this section, we evaluate and analyze the performance of FHDnn and ResNet-18 under several unreliable communication channels, packet loss, noise aggregation and bit errors respectively. To ensure a fair comparison, we employ the same hyperparameters for both models and evaluate them on the CIFAR-10 dataset. The detailed experimental settings are as follows: local training rounds $E = 2$, sampling ratio $C = 0.2$, batch size $B = 10$. The packet loss rate is set from 0 to 0.6, the signal-to-noise ratio (SNR) ranges from -10 to 25, and the bit error rate (BER) varies from 10^{-19} to 1. The detailed experimental results are presented as follows.

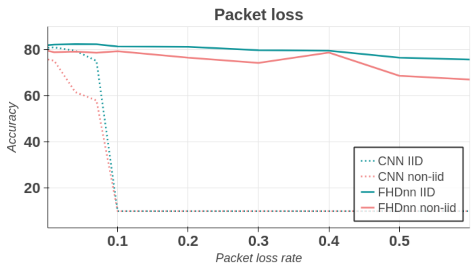


Fig. 14. Accuracy comparison of FHDnn with ResNet with packet loss conditions

1). **Packet Loss:** As shown in Fig. 14, if the packet loss rate is extremely small, e.g., below 10^{-2}, ResNet has very minimal accuracy loss. However, for more, realistic packet loss rates such as 20% the CNN model fails to converge. When there is packet loss, the central server replaces the model weights from the lost packets with zero values. For example, 20% packet loss rate implies 20% of the weights are zero. Moreover, this loss is accumulative as the models are averaged during each round of communication thereby giving the CNNs no chance of recovery. In contrast, FHDnn is highly robust to packet loss with almost no loss in accuracy. For FHDnn, since the data is distributed uniformly across the entire hypervector, a small amount of missing data is tolerable. However, since CNNs have a more structured representation of data with interconnections between neurons, the loss of weights affects the performance of subsequent layers which is detrimental to its performance.

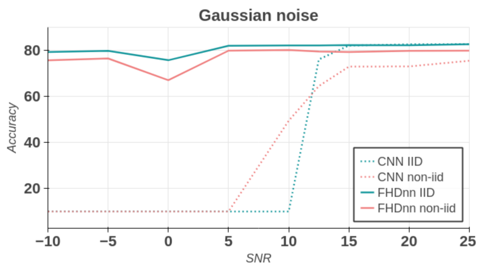


Fig. 15. Accuracy comparison of FHDnn with ResNet with gaussian noise conditions

2). **Gaussian Noise:** We experiment with different SNR to simulate noisy links, illustrated in Fig. 15. For higher SNRs, e.g., 25, the accuracy of the CNN model decreases by approximately 8%, while the accuracy of the FHDnn remains almost unchanged. When the SNR is less than 10, the CNN model fails completely, both in IID and non-IID settings. Its accuracy drops to 10%, which is not even as good as the performance of random classification. In contrast, the FHDnn model maintains stability and excellent accuracy even under extreme noise conditions. In particular, non-IID data usually introduces additional variability, but the degradation of FHDnn's performance in non-IID scenarios is also small. This further demonstrates that the FHDnn model has excellent robustness to noise and maintains high accuracy even at low SNR values where the CNN model fails completely, making it well suited for the noisy environments often encountered in real-world IoT and federated learning applications.

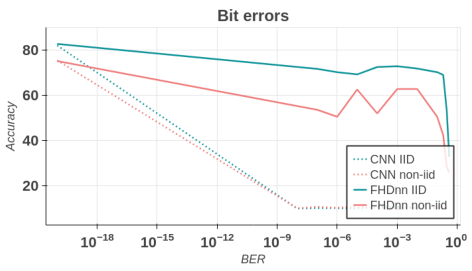


Fig. 16. Accuracy comparison of FHDnn with ResNet with bit errors conditions

3). **Bit Errors:** As shown in Fig. 16, the CNN accuracy drops much faster than the FHDnn when a bit error occurs. At 10^{-}9 BER conditions, it's accuracy drops drastically to the level of a random classification. This is due to the fact that the weights of CNN are floating point numbers and a single bit flip can significantly change the weights. In contrast, FHDnn can have a significantly better ability to withstand the effects of bit errors. At 10^{-9} BER, the accuracy for IID data is still close to 80%. All models fail when the BER reaches extreme levels close to 1. This is because the BER dominates communication and corrupts learning. However, before this catastrophic drop, FHDnn shows greater stability over a wider range of BERs. This is due to the fact that FHDnn uses an integer representation, which is to some extent going to be more tolerant of bit errors, and the quantization method we designed in section 5.3 with scaling further mitigates the errors.

6.6 Performance of proposed Communication Efficiency Strategies

To evaluate the strategies we design in the Section 5.4 to improve the communication efficiency of FHDnn, we conduct extensive experiments. Table 5 summary the final accuracy after 100 rounds of training and the improvement in communication cost for the respective approaches on MNIST dataset.

Table 5. Performance of Communication Efficiency Strategies

Method	Final Accuracy	Improvement
Baseline	94.1%	-
Binarized Differential Transmission	91.2%	32x
50% Subsampling	91.1%	2x
50% Sparsification & Compression	90.0%	2x
10% Subsampling	90.7%	10x
90% Sparsification & Compression	91.6%	10x

The baseline model achieves a final accuracy of 94.1%, with no communication improvement (compression factor = 1×). Binarized differential transmission method achieves a final accuracy of 91.2%, showing a marginal drop of 2.9% compared to the baseline. However, it provides a substantial 32× improvement in communication efficiency, making it highly effective for reducing communication overhead with minimal accuracy loss. For subsampling and sparsification & compression approaches, the communication improvement depends on the percentage of the model values that are subsampled or sparsified. For example, if we subsample 10% of the model, than the communication cost is reduced by 10x. Or, if we sparsify the model by 90%, the reduction is 10× again. We present the final accuracy and improvement in communication cost at different subsampling and sparsification percentages. The accuracy of all methods remains above 90%, which indicates that the several strategies we devised to improve communication efficiency are very effective and have very limited impact on model performance. After the optimization of the communication strategy, the communication cost of FHDnn is reduced by 2112× compared with ResNet-18, and the local computation and energy consumption are reduced by 48-192×.

7 CONCLUSION

In this paper, we introduce a lightweight, communication-efficient and highly robust federated learning framework, FHDnn. It first extracts complex features from raw data using the comparative learning framework SimCLR, and then maps them into high-dimensional vectors and updates the federated learning model using the HDC approach. This strategy speeds up the learning, reduces the transmission cost, and can achieve high accuracy on complex image tasks. Secondly, we analyze and prove in detail the complexity of performing HDC in a generalized federated learning framework, providing important theoretical guarantees. Finally, we design several more efficient communication strategies to improve the communication efficiency of FHDnn by 32×. We evaluate FHDnn through extensive experiments. The results show that FHDnn improves the convergence speed by 3×, reduces the communication cost by 2112×, and reduces the local client computation and energy consumption by 192× compared to the CNN-based federated learning approach. In addition, it has better robustness to unreliable communication setups including bit errors, noise, and packet loss.

ACKNOWLEDGEMENTS

This work was supported in part by CRISP, PRISM and CoCoSys, centers in JUMP1.0 and 2.0 (SRC programs sponsored by DARPA), SRC Global Research Collaboration grants (GRC TASK 3021.001), and NSF grants #2112665, #2003279, #2120019, #2211386, #2052809, #1911095, and #2112167.

A PROOF OF THEOREM 1

A.1 Additional Notation

Let $\mathbf{w}_t^k$ be the model maintained on the k-th device at the t-th step, and $\mathbf{w}_t$ be the global model. The clients communicate after E local epochs for global aggregation. Let $\mathcal{I}_E$ be the set of those aggregation steps, i.e., $\mathcal{I}_E = \{nE \mid n = 1, 2, ...\}$, so the client models are aggregated if $t + 1 \in \mathcal{I}_E$. We introduce an additional variable $\mathbf{v}_{t+1}^k$ to represent the result of the SGD steps where no communication occurs, similar to [42, 51]. The following equation describes the local updates of the clients:

$$\mathbf{v}_{t+1}^k = \mathbf{w}_t^k - \eta_t \nabla F_k(\mathbf{w}_t^k, \xi_t^k)$$

If $t + 1 \notin \mathcal{I}_E$, we have $\mathbf{w}_{t+1}^k = \mathbf{v}_{t+1}^k$ because there is no communication and aggregation of models. On the other hand, if $t + 1 \in \mathcal{I}_E$, the randomly selected clients $k \in \mathcal{S}_{t+1}$ communicate their models which are then aggregated and averaged at the server. This is summarized by the equation below.

$$\mathbf{w}_{t+1}^k = \begin{cases} \mathbf{v}_{t+1}^k, & \text{if } t + 1 \notin \mathcal{I}_E \\ \frac{1}{|\mathcal{S}_{t+1}|} \sum_{k \in \mathcal{S}_{t+1}} \mathbf{v}_{t+1}^k, & \text{if } t + 1 \in \mathcal{I}_E \end{cases}$$

The variable $\mathbf{w}_{t+1}^k$ can be interpreted as the model obtained directly after the communication steps. We define two virtual sequences $\overline{\mathbf{v}}_{t+1} = \sum_{k=1}^N p_k \mathbf{v}_t^k$ and $\overline{\mathbf{w}}_{t+1} = \sum_{k=1}^N p_k \mathbf{w}_t^k$. Notice that if we apply a single step of SGD to $\overline{\mathbf{w}}$, we get $\overline{\mathbf{v}}_{t+1}$. These sequences are denoted virtual because both are not available when $t + 1 \notin \mathcal{I}_E$ and we can only access $\overline{\mathbf{w}}_{t+1}$ when $t + 1 \in \mathcal{I}_E$. We also define $\overline{\mathbf{g}}_t = \sum_{k=1}^{N} p_k \nabla F_k(\mathbf{w}_t^k)$ and $\mathbf{g}_t = \sum_{k=1}^{N} p_k \nabla F_k(\mathbf{w}_t^k, \xi_t^k)$ for convenience of notation. Therefore, $\overline{\mathbf{v}}_{t+1} = \overline{\mathbf{w}}_t - \eta_t \mathbf{g}_t$ and $\mathbb{E}\|\mathbf{g}_t\| = \overline{\mathbf{g}}_t$. There are two sources of randomness in the following analysis. One results from the stochastic gradients and the other is from the random sampling of devices. To distinguish them, we use the notation $\mathbb{E}_{\mathcal{S}_t}(\cdot)$ when we take expectation over the randomness of stochastic gradients.

A.2 Lemmas

We present the necessary lemmas that we use in the proof of Theorem 1. These lemmas are derived and established in [42], so we defer their proofs.

Lemma 1 (Result of One Step SGD). *Assume Properties 1 and 2 hold. If* $\eta_t \leq \frac{1}{4L}$, *we have*

$$\mathbb{E}\|\overline{\mathbf{v}}_{t+1} - \mathbf{w}^*\|^2 \leq (1 - \eta_t \mu)\mathbb{E}\|\overline{\mathbf{w}}_t - \mathbf{w}^*\|^2$$
$$+\eta_t^2 \mathbb{E}\|\mathbf{g}_t - \overline{\mathbf{g}}_t\|^2 + 6L\eta_t^2 \Gamma + 2\mathbb{E}\Big[\sum_{k=1}^{N} p_k \|\overline{\mathbf{w}}_t - \mathbf{w}_t^k\|^2\Big].$$

Lemma 2 (Bounding the Variance). *Assume Property 3 holds. It follows that*

$$\mathbb{E}\|\mathbf{g}_t - \overline{\mathbf{g}}_t\|^2 \leq \sum_{k=1}^{N} p_k^2 \sigma_k^2.$$

Lemma 3 (Bounding the Divergence of $\mathbf{w}_t^k$). *Assume Property 4 holds,* η_t *is non-increasing and* $\eta_t \leq 2\eta_{t+E}$ *for all* $t \leq 0$. *It follows that*

$$\mathbb{E}\Big[\frac{1}{N}\sum_{k=1}^{N} \|\overline{\mathbf{w}}_t - \mathbf{w}_t^k\|^2\Big] \leq 4\eta_t^2 (E-1)^2 G^2.$$

Lemma 4 (Unbiased Sampling). *If* $t + 1 \in \mathcal{I}_E$, *we have*

$$\mathbb{E}_{\mathcal{S}_t}[\overline{\mathbf{w}}_{t+1}] = \overline{\mathbf{v}}_{t+1}$$

Lemma 5 (Bounding the Variance of $\mathbf{w}_t^k$). *For* $t + 1 \in \mathcal{I}_E$, *assume that* $\eta_t \leq 2\eta_{t+E}$ *for all* $t \geq 0$. *We then have*

$$\mathbb{E}_{\mathcal{S}_t}\|\overline{\mathbf{v}}_{t+1} - \overline{\mathbf{w}}_{t+1}\|^2 \leq \frac{N-K}{N-1}\frac{4}{K}\eta_t^2 E^2 G^2$$

A.3 Theorem 1

We have $\overline{\mathbf{w}}_{t+1} = \overline{\mathbf{v}}_{t+1}$ whether $t + 1 \in \mathcal{I}_E$ or $t + 1 \notin \mathcal{I}_E$. Then, we take the expectation over the randomness of stochastic gradient and use Lemma 1, Lemma 2, and Lemma 3 to get

$$\begin{aligned}
&\mathbb{E}\|\overline{\mathbf{w}}_{t+1} - \mathbf{w}^*\|^2 = \mathbb{E}\|\overline{\mathbf{v}}_{t+1} - \mathbf{w}^*\|^2 \\
&\leq (1 - \eta_t \mu)\mathbb{E}\|\overline{\mathbf{w}}_t - \mathbf{w}^*\|^2 + \eta_t^2 \mathbb{E}\|\mathbf{g}_t - \overline{\mathbf{g}}_t\|^2 \\
&\quad + 6L\eta_t^2 \Gamma + 2\mathbb{E}\Big[\sum_{k=1}^{N} p_k \|\overline{\mathbf{w}}_t - \mathbf{w}_t^k\|^2\Big] \\
&\leq (1 - \eta_t \mu)\mathbb{E}\|\overline{\mathbf{w}}_t - \mathbf{w}^*\|^2 \\
&\quad + \eta_t^2 \Big[\sum_{k=1}^{N} p_k^2 \sigma_k^2 + 6L\Gamma + 8(E-1)^2 G^2\Big].
\end{aligned}$$

If $t+1 \in \mathcal{I}_E$, note that

$$\begin{aligned}\|\overline{\mathbf{w}}_{t+1} - \mathbf{w}^*\|^2 &= \|\overline{\mathbf{w}}_{t+1} - \overline{\mathbf{v}}_{t+1} + \overline{\mathbf{v}}_{t+1} - \mathbf{w}^*\|^2 \\ &= \underbrace{\|\overline{\mathbf{w}}_{t+1} - \overline{\mathbf{v}}_{t+1}\|^2}_{A_1} + \underbrace{\|\overline{\mathbf{v}}_{t+1} - \mathbf{w}^*\|^2}_{A_2} \\ &+ \underbrace{\langle \overline{\mathbf{w}}_{t+1} - \overline{\mathbf{v}}_{t+1}, \overline{\mathbf{v}}_{t+1} - \mathbf{w}^* \rangle}_{A_3}.\end{aligned}$$

When the expectation is taken over $\mathcal{S}_{t+1}$, the term A_3 vanishes because $\mathbb{E}_{\mathcal{S}_{t+1}}[\overline{\mathbf{w}}_{t+1} - \overline{\mathbf{v}}_{t+1}] = 0$, that is, $\overline{\mathbf{w}}_{t+1}$ is unbiased. If $t+1 \notin \mathcal{I}_E$, A_1 is vanished as $\overline{\mathbf{w}}_{t+1} = \overline{\mathbf{v}}_{t+1}$. A_2 can be bounded using Lemmas 1 to 3 and Lemma 5. It follows that

$$\begin{aligned}\mathbb{E}\|\overline{\mathbf{w}}_{t+1} - \mathbf{w}^*\|^2 &\leq (1-\eta_t\mu)\mathbb{E}\|\overline{\mathbf{w}}_t - \mathbf{w}^*\|^2 \\ &+ \eta_t^2\Big[\sum_{k=1}^{N} p_k^2\sigma_k^2 + 6L\Gamma + 8(E-1)^2G^2\Big].\end{aligned}$$

If $t+1 \in \mathcal{I}_E$, the term A_1 can be additionally bounded using Lemma 5, then

$$\begin{aligned}&\mathbb{E}\|\overline{\mathbf{w}}_{t+1} - \overline{\mathbf{v}}_{t+1}\|^2 + \mathbb{E}\|\overline{\mathbf{v}}_{t+1} - \mathbf{w}^*\|^2 \\ &\leq (1-\eta_t\mu)\mathbb{E}\|\overline{\mathbf{w}}_t - \mathbf{w}^*\|^2 \\ &+ \eta_t^2\Big[\sum_{k=1}^{N} p_k^2\sigma_k^2 + 6L\Gamma + 8(E-1)^2G^2 + \frac{N-K}{N-1}\frac{4}{K}E^2G^2\Big].\end{aligned}$$

Now, let $\Delta_t = \|\overline{\mathbf{w}}_{t+1} - \mathbf{w}^*\|^2$ for notational convenience. Also, let

$$B = \sum_{k=1}^{N} p_k^2\sigma_k^2 + 6L\Gamma + 8(E-1)^2G^2 + \frac{N-K}{N-1}\frac{4}{K}E^2G^2.$$

We use a diminishing learning rate with $\eta_t = \frac{\beta}{t+\gamma}$ for some $\beta \geq \frac{1}{\mu}$ and $\gamma > 0$ such that $\eta_1 \leq \min\{\frac{1}{\mu}, \frac{1}{4L}\} = \frac{1}{4L}$ and $\eta_t \leq 2\eta_{t+E}$. Now, we prove by induction that

$$\Delta_t \leq \frac{v}{\gamma + t}$$

where

$$v = \max\{\frac{\beta^2 B}{\beta\mu - 1}, (\gamma+1)\Delta_0\}.$$

The definition of v ensures that it holds for $t = 0$. If we assume the result holds for some $t > 0$, it follows

$$\begin{aligned}\Delta_{t+1} &\leq (1-\eta_t\mu)\Delta_t + \eta_t^2 B \\ &= \Big(1 - \frac{\beta\mu}{t+\gamma}\Big)\frac{v}{t+\gamma} + \frac{\beta^2 B}{(t+\gamma)^2} \\ &= \frac{t+\gamma-1}{(t+\gamma)^2}v + \Big[\frac{\beta^2 B}{(t+\gamma)^2} - \frac{\beta\mu - 1}{(t+\gamma)^2}v\Big] \\ &\leq \frac{v}{t+\gamma+1}.\end{aligned}$$

Then, by the strong convexity of $F(\cdot)$,

$$\mathbb{E}[F(\overline{\mathbf{w}}_t)] - F^* \leq \frac{L}{2}\Delta_t \leq \frac{L}{2}\frac{v}{\gamma + t}.$$

If we choose the parameters as $\beta = \frac{2}{\mu}$, $\gamma = \max\{8\frac{L}{\mu} - 1, E\}$ and define $\kappa = \frac{L}{\mu}$, then $\eta_t = \frac{2}{\mu}\frac{1}{\gamma+t}$. Using the fact that $\max\{a, b\} \leq a + b$, we have

$$\begin{aligned} v &\leq \frac{\beta^2 B}{\beta\mu - 1} + (\gamma + 1)\Delta_0 \\ &= 4\frac{B}{\mu^2} + (\gamma + 1)\Delta_0 \\ &\leq 4\frac{B}{\mu^2} + \left(8\frac{L}{\mu} - 1 + E + 1\right)\Delta_0 \\ &= 4\frac{B}{\mu^2} + \left(8\frac{L}{\mu} + E\right)\|\overline{\mathbf{w}}_1 - \mathbf{w}^*\|^2. \end{aligned}$$

Therefore,

$$\begin{aligned} \mathbb{E}[F(\overline{\mathbf{w}}_t)] - F^* &\leq \frac{L}{2(\gamma + t)}\left[4\frac{B}{\mu^2} + \left(8\frac{L}{\mu} + E\right)\|\overline{\mathbf{w}}_1 - \mathbf{w}^*\|^2\right] \\ &= \frac{2\kappa}{\gamma + t}\left[\frac{B}{\mu} + \left(2L + \frac{E\mu}{4}\right)\|\overline{\mathbf{w}}_1 - \mathbf{w}^*\|^2\right]. \end{aligned} \tag{28}$$

REFERENCES

[1] [n. d.]. UCI machine learning repository. http://archive.ics.uci.edu/ml/datasets/ISOLET.

[2] Davide Anguita, Alessandro Ghio, Luca Oneto, Xavier Parra, Jorge Luis Reyes-Ortiz, et al. 2013. A public domain dataset for human activity recognition using smartphones.. In *Esann*, Vol. 3. 3.

[3] Ravikumar Balakrishnan, Mustafa Akdeniz, Sagar Dhakal, and Nageen Himayat. 2020. Resource management and fairness for federated learning over wireless edge networks. In *2020 IEEE 21st International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*. IEEE, 1–5.

[4] Jeremy Bernstein, Yu-Xiang Wang, Kamyar Azizzadenesheli, and Animashree Anandkumar. 2018. signSGD: Compressed optimisation for non-convex problems. In *International Conference on Machine Learning*. PMLR, 560–569.

[5] Sebastian Caldas, Jakub Konečny, H Brendan McMahan, and Ameet Talwalkar. 2018. Expanding the reach of federated learning by reducing client resource requirements. *arXiv preprint arXiv:1812.07210* (2018).

[6] Rishikanth Chandrasekaran, Kazim Ergun, Jihyun Lee, Dhanush Nanjunda, Jaeyoung Kang, and Tajana Rosing. 2022. Fhdnn: Communication efficient and robust federated learning for aiot networks. In *Proceedings of the 59th ACM/IEEE Design Automation Conference*. 37–42.

[7] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. 2020. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*. PMLR, 1597–1607.

[8] Wenlin Chen, Samuel Horvath, and Peter Richtarik. 2020. Optimal client sampling for federated learning. *arXiv preprint arXiv:2010.13723* (2020).

[9] NVIDIA Corporation. [n. d.]. Nvidia Jetson. https://developer.nvidia.com/embedded/jetson-nano-developer-kit.

[10] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. 2015. Binaryconnect: Training deep neural networks with binary weights during propagations. *Advances in neural information processing systems* 28 (2015).

[11] Li Deng. 2012. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine* 29, 6 (2012), 141–142.

[12] Arpan Dutta, Saransh Gupta, Behnam Khaleghi, Rishikanth Chandrasekaran, Weihong Xu, and Tajana Rosing. 2022. HDnn-PIM: Efficient in Memory Design of Hyperdimensional Computing with Feature Extraction. In *Proceedings of the Great Lakes Symposium on VLSI 2022*. 281–286.

[13] Ronald A Fisher. 1936. The use of multiple measurements in taxonomic problems. *Annals of eugenics* 7, 2 (1936), 179–188.

[14] Stephen I Gallant et al. 1990. Perceptron-based learning algorithms. *IEEE Transactions on neural networks* 1, 2 (1990), 179–191.

[15] Michael Gastpar. 2008. Uncoded transmission is exactly optimal for a simple Gaussian "sensor" network. *IEEE Transactions on Information Theory* 54, 11 (2008), 5247–5251.

[16] Lulu Ge and Keshab K Parhi. 2020. Classification using hyperdimensional computing: A review. *IEEE Circuits and Systems Magazine* 20, 2 (2020), 30–47.

[17] Andrea Goldsmith. 2005. *Wireless communications.* Cambridge university press.

[18] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition.* 770–778.

[19] Samuel Horváth, Chen-Yu Ho, Ludovit Horvath, Atal Narayan Sahu, Marco Canini, and Peter Richtárik. 2019. Natural compression for distributed deep learning. *arXiv preprint arXiv:1905.10988* (2019).

[20] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861* (2017).

[21] Cheng-Yen Hsieh, Yu-Chuan Chuang, and An-Yeu Andy Wu. 2021. Fl-hdc: Hyperdimensional computing design for the application of federated learning. In *2021 IEEE 3rd International Conference on Artificial Intelligence Circuits and Systems (AICAS).* IEEE, 1–5.

[22] IDC. 2019. The Growth in Connected IoT Devices. [Online].

[23] Mohsen Imani et al. [n. d.]. Bric: Locality-based encoding for energy-efficient brain-inspired hyperdimensional computing. In *Proceedings of the 56th Annual Design Automation Conference 2019.*

[24] Mohsen Imani, Yeseong Kim, Sadegh Riazi, John Messerly, Patric Liu, Farinaz Koushanfar, and Tajana Rosing. 2019. A framework for collaborative learning in secure high-dimensional space. In *2019 IEEE 12th International Conference on Cloud Computing (CLOUD).* IEEE, 435–446.

[25] Mohsen Imani, Deqian Kong, Abbas Rahimi, and Tajana Rosing. 2017. Voicehd: Hyperdimensional computing for efficient speech recognition. In *2017 IEEE international conference on rebooting computing (ICRC).* IEEE, 1–8.

[26] Yuang Jiang, Shiqiang Wang, Victor Valls, Bong Jun Ko, Wei-Han Lee, Kin K Leung, and Leandros Tassiulas. 2022. Model pruning enables efficient federated learning on edge devices. *IEEE Transactions on Neural Networks and Learning Systems* (2022).

[27] Pentti Kanerva. 2009. Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors. *Cognitive computation* 1, 2 (2009), 139–159.

[28] Pentii Kanerva, Jan Kristoferson, and Anders Holst. 2000. Random indexing of text samples for latent semantic analysis. In *Proceedings of the Annual Meeting of the Cognitive Science Society*, Vol. 22.

[29] Sai Praneeth Karimireddy, Satyen Kale, Mehryar Mohri, Sashank Reddi, Sebastian Stich, and Ananda Theertha Suresh. 2020. Scaffold: Stochastic controlled averaging for federated learning. In *International Conference on Machine Learning.* PMLR, 5132–5143.

[30] Harsh Kasyap, Somanath Tripathy, and Mauro Conti. 2023. HDFL: Private and Robust Federated Learning using Hyperdimensional Computing. In *2023 IEEE 22nd International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom).* IEEE, 214–221.

[31] Behnam Khaleghi, Jaeyoung Kang, Hanyang Xu, Justin Morris, and Tajana Rosing. 2022. GENERIC: highly efficient learning engine on edge using hyperdimensional computing. In *Proceedings of the 59th ACM/IEEE Design Automation Conference.* 1117–1122.

[32] Yeseong Kim, Mohsen Imani, and Tajana S Rosing. 2018. Efficient human activity recognition using hyperdimensional computing. In *Proceedings of the 8th International Conference on the Internet of Things.* 1–6.

[33] Jakub Konečnỳ, H Brendan McMahan, Daniel Ramage, and Peter Richtárik. 2016. Federated optimization: Distributed machine learning for on-device intelligence. *arXiv preprint arXiv:1610.02527* (2016).

[34] Jakub Konečnỳ, H Brendan McMahan, Felix X Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. 2016. Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492* (2016).

[35] Jakub Konečnỳ and Peter Richtárik. 2018. Randomized distributed mean estimation: Accuracy vs. communication. *Frontiers in Applied Mathematics and Statistics* (2018), 62.

[36] Alex Krizhevsky, Geoffrey Hinton, et al. 2009. Learning multiple layers of features from tiny images. (2009).

[37] James F Kurose. 2005. *Computer networking: A top-down approach featuring the internet, 3/E.* Pearson Education India.

[38] Yann LeCun *et al.* 2015. Deep learning. *nature* (2015).

[39] Fei-Fei Li, Marco Andreeto, Marc'Aurelio Ranzato, and Pietro Perona. 2022. Caltech 101. https://doi.org/10.22002/D1.20086

[40] Haomin Li, Fangxin Liu, Yichi Chen, and Li Jiang. 2024. HyperFeel: An Efficient Federated Learning Framework Using Hyperdimensional Computing. In *2024 29th Asia and South Pacific Design Automation Conference (ASP-DAC).* IEEE, 716–721.

[41] Li Li, Yuxi Fan, Mike Tse, and Kuo-Yi Lin. 2020. A review of applications in federated learning. *Computers & Industrial Engineering* 149 (2020), 106854.

[42] Xiang Li *et al.* 2019. On the convergence of fedavg on non-iid data. *arXiv preprint arXiv:1907.02189* (2019).

[43] Liqiang Lu and Yun Liang. 2018. SpWA: An efficient sparse winograd convolutional neural networks accelerator on FPGAs. In *Proceedings of the 55th Annual Design Automation Conference*. 1–6.

[44] Brendan McMahan *et al.* 2017. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*.

[45] Deniz Oktay, Johannes Ballé, Saurabh Singh, and Abhinav Shrivastava. 2019. Scalable model compression by entropy penalized reparameterization. *arXiv preprint arXiv:1906.06624* (2019).

[46] Raspberry Pi. [n. d.]. Raspberry Pi 4. https://www.raspberrypi.com/products/raspberry-pi-4-model-b/.

[47] Abbas Rahimi et al. 2018. Efficient biosignal processing using hyperdimensional computing: Network templates for combined learning and classification of ExG signals. *Proc. IEEE* 107, 1 (2018), 123–143.

[48] Abbas Rahimi, Pentti Kanerva, and Jan M Rabaey. 2016. A robust and energy-efficient classifier using brain-inspired hyperdimensional computing. In *Proceedings of the 2016 international symposium on low power electronics and design*. 64–69.

[49] Amirhossein Reisizadeh, Aryan Mokhtari, Hamed Hassani, Ali Jadbabaie, and Ramtin Pedarsani. 2020. Fedpaq: A communication-efficient federated learning method with periodic averaging and quantization. In *International Conference on Artificial Intelligence and Statistics*. PMLR, 2021–2031.

[50] Antonello Rosato, Massimo Panella, Evgeny Osipov, and Denis Kleyko. 2022. Few-shot federated learning in randomized neural networks via hyperdimensional computing. In *2022 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 1–8.

[51] Sebastian U Stich. 2018. Local SGD converges fast and communicates little. *arXiv preprint arXiv:1805.09767* (2018).

[52] Ananda Theertha Suresh, X Yu Felix, Sanjiv Kumar, and H Brendan McMahan. 2017. Distributed mean estimation with limited communication. In *International Conference on Machine Learning*. PMLR, 3329–3337.

[53] Chandra Thapa, Mahawaga Arachchige Pathum Chamikara, Seyit Camtepe, and Lichao Sun. 2020. Splitfed: When federated learning meets split learning. *arXiv preprint arXiv:2004.12088* (2020).

[54] Anthony Thomas, Sanjoy Dasgupta, and Tajana Rosing. 2021. Theoretical Foundations of Hyperdimensional Computing. *Journal of Artificial Intelligence Research* 72 (2021), 215–249.

[55] Hui-Po Wang, Sebastian U Stich, Yang He, and Mario Fritz. 2021. ProgFed: Effective, Communication, and Computation Efficient Federated Learning by Progressive Training. *arXiv preprint arXiv:2110.05323* (2021).

[56] Yao Wang, Stephan Wenger, Jiantao Wen, and Aggelos K Katsaggelos. 2000. Error resilient video coding techniques. *IEEE signal processing magazine* 17, 4 (2000), 61–82.

[57] Jianqiao Wangni, Jialei Wang, Ji Liu, and Tong Zhang. 2018. Gradient sparsification for communication-efficient distributed optimization. *Advances in Neural Information Processing Systems* 31 (2018).

[58] Xizixiang Wei and Cong Shen. 2022. Federated learning over noisy channels: Convergence analysis and design examples. *IEEE Transactions on Cognitive Communications and Networking* (2022).

[59] Jie Wen, Zhixia Zhang, Yang Lan, Zhihua Cui, Jianghui Cai, and Wensheng Zhang. 2023. A survey on federated learning: challenges and applications. *International Journal of Machine Learning and Cybernetics* 14, 2 (2023), 513–535.

[60] Han Xiao, Kashif Rasul, and Roland Vollgraf. 2017. *Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms.* arXiv:cs.LG/1708.07747 [cs.LG]

[61] Zirui Xu, Fuxun Yu, Jinjun Xiong, and Xiang Chen. 2021. Helios: heterogeneity-aware federated learning with dynamically balanced collaboration. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 997–1002.

[62] Qiang Yang, Yang Liu, Yong Cheng, Yan Kang, Tianjian Chen, and Han Yu. 2020. Horizontal federated learning. In *Federated Learning*. Springer, 49–67.

[63] Nikita Zeulin, Olga Galinina, Nageen Himayat, and Sergey Andreev. 2023. Resource-Efficient Federated Hyperdimensional Computing. *arXiv preprint arXiv:2306.01339* (2023).

[64] Quanling Zhao, Kai Lee, Jeffrey Liu, Muhammad Huzaifa, Xiaofan Yu, and Tajana Rosing. 2022. Fedhd: federated learning with hyperdimensional computing. In *Proceedings of the 28th Annual International Conference on Mobile Computing And Networking*. 791–793.

[65] Quanling Zhao, Xiaofan Yu, Shengfan Hu, and Tajana Rosing. 2024. MultimodalHD: Federated Learning Over Heterogeneous Sensor Modalities using Hyperdimensional Computing. In *2024 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 1–6.

[66] Quanling Zhao, Xiaofan Yu, and Tajana Rosing. 2023. Attentive Multimodal Learning on Sensor Data using Hyperdimensional Computing. In *Proceedings of the 22nd International Conference on Information Processing in Sensor Networks*. 312–313.

[67] Michael Zhu and Suyog Gupta. 2017. To prune, or not to prune: exploring the efficacy of pruning for model compression. *arXiv preprint arXiv:1710.01878* (2017).