FHDnn: Communication Efficient and Robust Federated Learning for AIoT Networks

Rishikanth Chandrasekaran*, Kazim Ergun*, Jihyun Lee, Dhanush Nanjunda, Jaeyoung Kang
Tajana Rosing
Department of Computer Science and Engineering, UC San Diego, La Jolla, CA 92093
{r3chandr, kergun, h0l003, dnanjund, j5kang, tajana}@ucsd.edu

Abstract

The advent of IoT and advances in edge computing inspired federated learning, a distributed algorithm to enable on device learning. Transmission costs, unreliable networks and limited compute power all of which are typical characteristics of IoT networks pose a severe bottleneck for federated learning. In this work we propose FHDnn, a synergetic federated learning framework that combines the salient aspects of CNNs and Hyperdimensional Computing. FHDnn performs hyperdimensional learning on features extracted from a self-supervised contrastive learning framework to accelerate training, lower communication costs, and increase robustness to network errors by avoiding the transmission of the CNN and training only the hyperdimensional component. Compared to CNNs, we show through experiments that FHDnn reduces communication costs by 66×, local client compute and energy consumption by 1.5 - 6×, while being highly robust to network errors with minimal loss in accuracy.

CCS Concepts

• **Computing methodologies** → **Machine learning**.

Keywords

Hyperdimensional Computing, Federated Learning

ACM Reference Format:
Rishikanth Chandrasekaran*, Kazim Ergun*, Jihyun Lee, Dhanush Nanjunda, Jaeyoung Kang, Tajana Rosing. 2022. FHDnn: Communication Efficient and Robust Federated Learning for AIoT Networks. In *Proceedings of the 59th ACM/IEEE Design Automation Conference (DAC) (DAC '22), July 10–14, 2022, San Francisco, CA, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3489517.3530394

1 Introduction

A group of distributed edge devices communicating with each other and sharing data is loosely termed the Internet of Things (IoT). These edge devices are privy to a rich source of data which when leveraged can enable various smart applications such as smart cities [15] [2] and AI-enabled farming [18]. However, often the private and sensitive nature of the data coupled with high transmission costs prevent the central aggregation of data to the cloud. Recent advances in edge computing enabled the idea of distributed computing for on device processing. One such distributed learning paradigm is federated learning (FL) [17]. FL learns a machine learning model on data distributed across various devices without having to aggregate them centrally. FL works by training models locally on the device with data visible to each device and then averages these models from all participating devices.

*Both authors contributed equally to this research

DAC '22, July 10–14, 2022, San Francisco, CA, USA

ACM ISBN 978-1-4503-9142-9/22/07.
https://doi.org/10.1145/3489517.3530394

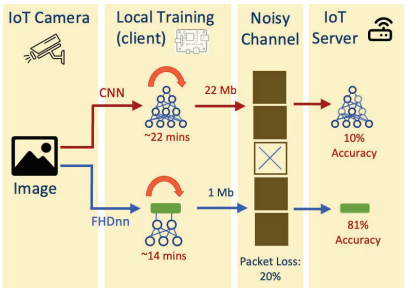

Figure 1: FHDnn against CNNs for federated learning

Transmission costs, unreliable networks, and limited on device computer power pose significant challenges for FL. Previous works [4, 5] have explored model compression methods and dropout techniques to reduce the communication cost by decreasing the size of the model updates. However, these methods do not factor in the non-ideality of IoT networks, assuming reliable lossless communication and subsequently are neither robust to network errors nor provide guarantees for convergence. Also, IoT often uses Low Power Wide Area Networks (LP-WAN) to conserve energy of battery operated edge devicesm but it has limited bandwidth, high packet loss and no sophisticated coding scheme making FL vulnerable to errors.

We present FHDnn a novel synergetic federated learning framework that combines 2 different learning paradigms of Deep Learning and Hyperdimensional Computing (HDC) [11]. Deep learning excels at learning a complex hierarchy of features and boasts high accuracy however at the cost of compute power often requiring GPUs to train. HDC on the other hand features lightweight training using simple operations on distributed low precision representations that are inherently robust to errors. However, they don't enjoy the same accuracy as deep learning due to their inability to learn features automatically. FHDnn combines the complimentary salient features of both learning methodologies to enable a lightweight highly robust FL framework that addresses each of the above challenges. In this work, we limit ourselves to the problem of image classification, a common application in IoT.

FHDnn uses a pre-trained Convolutional Neural Network (CNN) as a feature extractor, the output of which are encoded into hypervectors that are then used for training a federated HD learner. Specifically we utilize a CNN trained using SimCLR [6] a contrastive learning framework which learns informative representations of images in a self-supervised manner that generalizes well to several datasets. FHDnn avoids the transmission of the CNN and instead trains only the HD classifier in a federated manner. This simple strategy accelerates learning, reduces transmission cost and utilizes the inherent robustness of HDC to tackle network errors as shown in Figure 1. In this work, we detail the architecture of FHDnn and systematically compare the performance of FHDnn with CNN under various settings. We summarize our key contributions below:

- We propose FHDnn, a novel FL strategy that is robust to lossy network transmission, is incredibly lightweight to train, and converges faster.

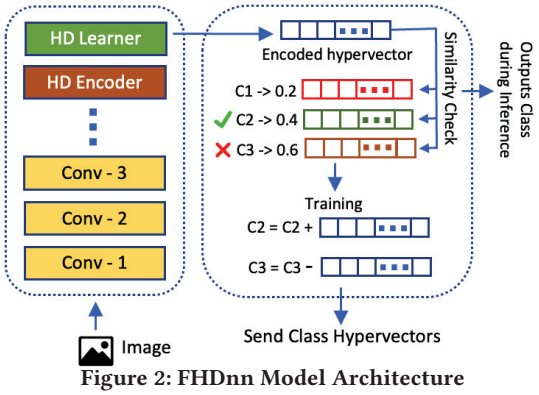

Figure 2: FHDnn Model Architecture

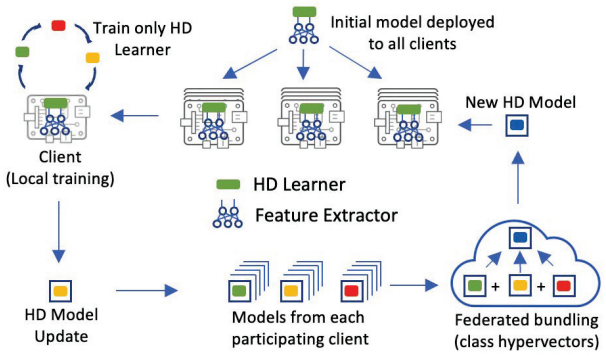

Figure 3: FHDnn Federated Training

- We empirically show through numerous experiments that FHDnn is robust to lossy network conditions. Specifically we evaluate FHDnn under three different unreliable network settings: packet loss, noise injection, and bit errors.
- We show that our approach reduces the communication costs by 66×, and reduces the local computations cost on the clients by up to 6× while being robust to lossy transmissions.

2 Background and Motivation

2.1 IoT networks

IoT networks often involve a large number of battery operated edge devices operating on a Low Power Wide Area Networks (LPWAN). LPWAN networks have limited bandwidth, narrow spectrum, and often lack any advanced modulation schemes due to compute cost and power constraints. Further, the network performance is worse due to the presence of packet loss which is highly prevalent in these networks [19, 20]. A study [16] shows that retransmission is non ideal as it further increases energy consumption and reduces network performance due to limited capacity. However, [9] shows that tolerating a packet loss rate of 20% allows for increased energy efficiency and network capacity.

2.2 Challenges in FL

Communication Efficiency: FL involves multiple rounds of communication typically until a target test accuracy is achieved. Each round, the participating clients send their models to the server. Complex models, like CNNs, contain millions of parameters resulting in large updates. FL typically takes several rounds to converge to the optimum which further exacerbates the communication cost.

Lossy Transmission: As detailed above, IoT networks are often unreliable which adds noise to the model updates causing convergence issues. CNNs in particular are not robust to noise on weights as shown in [3]. Failure to converge results in poor accuracy while longer convergence times results in increased communication costs.

Resource constraints: Battery operated edge devices typical to IoT networks have limited power and computation budgets. CNN based FL methods require edge devices to perform on-device back-propogation during each round of training which is computationally expensive incurring high resource usage.

3 Proposed Method: FHDnn

3.1 Model Architecture

Figure 2 shows the model architecture of FHDnn. In the following subsections we detail the 2 components of FHDnn: i) a pre-trained CNN as a feature extractor, ii) a federated HD learner, followed by the training methodology.

3.2 Feature extractor

While in theory any standard CNN can be used as a feature extractor, due to its salient characteristics we use a pre-trained SimCLR Resnet model as our feature extractor. SimCLR [6] 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 [6]) 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 better choice of feature extractor. Note that We choose the ResNet architecture due to availability of pre-trained models. One could use other models such as MobileNet, which are more ideal for edge devices with resource constraints.

3.3 HD learner

HDC is a computing paradigm based on biologically plausible models of data representation [12]. HD works by encoding data into low precision vectors of very large dimensions, referred to as hyper vectors in literature. HD classifiers operate on these vectors using binding and bundling operations which are simple and highly parallelizable.

Here we are concerned with encoding the output of a neural network. We use an encoding method proposed in [10] based on the notion of random projection. This approach embeds the data into a high-dimensional Euclidean space under a random linear map before quantizing them. More formally, given a point $x \in \mathcal{X}$, the features $z \subset \mathbb{Z}^n$ 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 $\phi(z) = \text{sign}(\Phi z)$ under the encoding function $\phi : \mathbb{Z} \to \mathcal{H}$ the rows of which $\Phi \in \mathbb{R}^{d \times n}$ are generated by randomly sampling directions from the n-dimensional unit sphere. sign(Φz) is the element-wise sign function returning +1 if $\Phi z \geq 0$ and −1 otherwise.

3.4 Federated Training

Figure 3 summarizes the federated training process for FHDnn.

3.4.1 Client Local Training: Each client initially starts with a feature extractor f and an untrained HD learner. Once we get the encoded hypervectors using the method described above, 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 Figure 2. We define the complete HD model $\mathbf{C}$ as the concatenation of class hypervectors, i.e., $\mathbf{C} = [\mathbf{c}_1^T, \mathbf{c}_2^T, ..., \mathbf{c}_n^T]$.

3.4.2 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 $n \in [N]$ sets its model $\mathbf{C}_t^n = \mathbf{C}_t$ and then performs training for E epochs using local data as described in 3.4.1
(3) **Aggregation:** The central server receives and aggregates the local models to produce a new global model:

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

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

3.5 FL Over Unreliable Channels With FHDnn

Federated learning is often carried out over wireless channels that attenuate the transmitted signal and introduce noise followed by packet losses. The centralized server is assumed to be able to broadcast the models reliably, error-free at arbitrary rates, which is a common assumption in many recent works. For uplink communications, the channel capacity per client is more constrained and unreliable due to shared wireless medium, even at very low rates.

The bandwidth allocated per client decreases with the number of clients, so does the capacity. Accordingly, the volume of data that can be conveyed reliably, i.e, throughput, scales by $1/N$. This implies that the data rates will be small, resulting in slow training speed unless transmission power is increased, which is undesirable considering energy consumption concerns.

Instead of limiting the rate to achieve error-free communication, we admit errors for the channel output at the server as perturbations in the client models can be tolerated to an extent by FHDnn. If the model is robust to errors, then there is no need for perfectly reliable transmissions. Thus, we analyze FHDnn assuming that the clients communicate over unreliable channels and the transmitted models are corrupted.

We consider three error models at different layers of the network stack. All models are applicable in practice depending on the underlying protocol. We first explore the properties of HD computing that makes the learning robust under the considered error models, then introduce different techniques for further improvement.

3.5.1 Noisy Aggregation. Conventional systems use source and channel coding to ensure reliability which are often unavailable in LPWAN networks. For noisy aggregation, as an alternative to the conventional pipeline, we assume uncoded transmission. This scheme bypasses the transformation of the model to a sequence of bits, which 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 (2), 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

$$\tilde{\mathbf{C}}_t^k = \mathbf{C}_t^k + \mathbf{n}_t^k \tag{2}$$

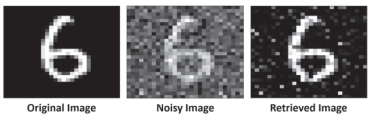

Figure 4: Noise robustness of hyperdimensional encodings

where $\mathbf{n}_t^k \sim \mathcal{N}(0, \sigma_{t,k}^2)$ is the $d \times n$-dimensional additive noise. Then, the signal-to-noise ratio (SNR) is:

$$SNR_{t,k} = \frac{\mathbb{E}\|\mathbf{C}_t^k\|^2}{\mathbb{E}\|\mathbf{n}_t^k\|^2} = \frac{P_{t,k}}{\sigma_{t,k}^2} \tag{3}$$

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:

$$SNR_t = \frac{\mathbb{E}\|\sum_{k=1}^N \mathbf{C}_t^k\|}{\mathbb{E}\|\sum_{k=1}^N \mathbf{n}_t^k\|} \approx \frac{N^2 P_{t,k}}{N\sigma_{t,k}^2} = N \times SNR_{t,k} \tag{4}$$

Notice that the effect of noise is suppressed by N times due to bundling. This claim can also be made for the FedAvg 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 a 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. Since the noise in each dimension can also be assumed to be independent, we can leverage the information spread to further eliminate noise.

Consider the random projection encoding described in Section 3.3. Let the encoding matrix $\Phi \in \mathbb{R}^{d \times n}$ expressed in terms of its d row vectors, i.e., $\Phi = [\Phi_1, \Phi_2, ..., \Phi_d]^T$. Then, the hypervector formed by encoding information $x \in \mathcal{X}$ can be written as $\mathbf{h} = [\Phi_1^T x, \Phi_2^T x, ..., \Phi_d^T x]^T$, where $x = [x_1, x_2, ..., x_n]^T$. As implied by this expression, the information is dispersed over the hypervectors uniformly. Now consider additive noise over the same hypervector such that $\mathbf{h} + \mathbf{n} = [\Phi_1^T x + n_1, \Phi_2^T x + n_2, ..., \Phi_d^T x + n_d]^T$. We can reconstruct the encoded information from the noisy hypervector $\tilde{\mathbf{h}} = \mathbf{h} + \mathbf{n}$ as follows:

$$x \approx \left[\frac{1}{d}\sum_{i=1}^{d} \Phi_{i,1}\tilde{\mathbf{h}}_i, \frac{1}{d}\sum_{i=1}^{d} \Phi_{i,2}\tilde{\mathbf{h}}_i, ..., \frac{1}{d}\sum_{i=1}^{d} \Phi_{i,n}\tilde{\mathbf{h}}_i\right] \tag{5}$$

where $\tilde{\mathbf{h}}_i = \Phi_i^T x + n_i$ are the elements of the noisy hypervector. The noise variance is then reduced by the averaging operation, similar to the case in Equation (4). 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. Figure 4 shows the original image, noisy image in the sample space, and reconstructed image for which the noise was added in the hyperdimensional space.

3.5.2 Bit Errors. We use bit error rate (BER) in conventional coded transmission as a figure of merit for system robustness. It is a measure on how accurately the receiver is able to decode transmitted data. The errors are bit flips in the received digital symbols, and are simply evaluated by the difference (usually Hamming distance) between the input bitstream of channel encoder and the output bitstream of channel decoder. Let $\hat{\mathbf{c}}$ be the binary coded model parameters that

are communicated to the server. For the bit error model, we treat the channel as a binary symmetric channel (BSC), which independently flips each bit in $\hat{\mathbf{c}}$ with probability p_e (e.g., $0 \rightarrow 1$). The received bitstream output at the server for client k at round t is then as follows:

$$\tilde{\hat{\mathbf{C}}}_t^k = \hat{\mathbf{C}}_t^k \oplus \mathbf{e}_t^k \tag{6}$$

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

$$\mathbb{P}(\mathbf{e}_t^k = \mathbf{v}) = p_e^{\mathrm{wt}(\mathbf{v})}(1-p_e)^{m-\mathrm{wt}(\mathbf{v})} \tag{7}$$

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 bitstream in (6) is finally reconstructed to a real-valued model, i.e., $\tilde{\hat{\mathbf{C}}}_t^k \rightarrow \tilde{\mathbf{C}}_t^k$.

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

A similar problem exists with HD model parameters, but to a lesser extent because the class prototypes use integer representations. Particularly, errors in the most significant bits (MSB) of integer representation leads to higher accuracy drop. We propose a quantizer solution to prevent this. 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 $\mathbf{c}_k,\ k \in \{1, ..., K\}$, 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.

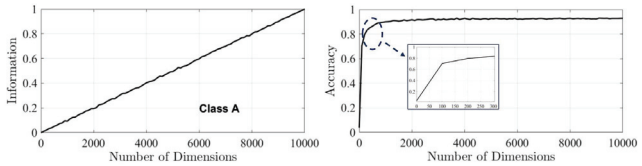

Figure 5: Impact of partial information on similarity check (left) and classification accuracy (right)

This way, bit errors are applied to the scaled up values. Intuitively, we limit the impact of the bit error on the models. Remember, from Section 3.4.1, that prediction is realized by a normalized dot-product between the encoded query and class hypervectors. 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.

3.5.3 Packet Loss. At the physical layer of the network stack, errors are observed in the form of additive noise or bit flips directly on the transmitted data. On the other hand, at the network and transport layers, packet losses are introduced. The combination of network and protocol specifications allows us to describe the error characteristics, with which the data transmission process has to cope.

The form of allowed errors, either bit errors or packet losses, are decided by the error control mechanism. For the previous error model, we assumed that the bit errors are admitted to propagate through the transport hierarchy. This assumption is valid for a family of protocols used in error resilient applications that can cope with such bit errors. In some protocols, the reaction of the system to any number of bit errors is to drop the corrupted packets. These protocols employ a cyclic redundancy check (CRC) or a checksum that allows the detection of bit errors. In such a case, the communication could assume bit-error free, but packet lossy link. We use the packet error rate (PER) metric as a performance measure, whose expectation is denoted packet error probability p_p. For a packet length of N_p bits, this probability can be expressed as:

$$p_p = 1 - (1-p_e)^{N_p} \tag{8}$$

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 unreliable and cannot guarantee packet delivery, but is low-latency and has much less overhead compared to TCP.

HD computing'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.

We demonstrate the property of obtaining partial information as an example using a speech recognition dataset [1]. In Figure 5(a), after training the model, we increasingly 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

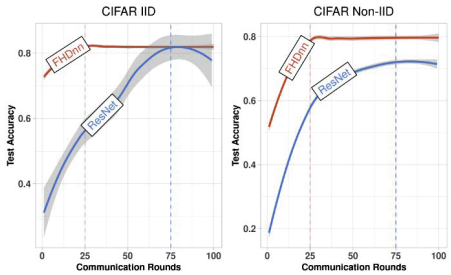

Figure 6: Accuracy and Number of communication rounds for various hyperparameters

dimensions. Figure 5(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 ∼ 90% accuracy even when 80% of the hypervector dimensions are removed.

3.6 Convergence Performance of FHDnn

It is commonly preferred to have smooth, strongly-convex, differentiable models to maintain a provable, analytically tractable convergence analysis for federated learning. Such models also provide faster convergence properties than the others. The learning computations in FHDnn 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, CNN-based approaches. FHDnn diverges from traditional linear methods in this respect. It enjoys both the superior performance properties of non-linear models and low computational complexity of linear models. This is a direct result of HD computing, which embeds data into a high-dimensional space where the geometry is such that simple learning methods are effective. The linearity in HD training benefits convergence, and at the same time the performance does not degrade due to the properties of non-linear hyperdimensional embeddings.

FHDnn, when posed as a distributed optimization solution, has the following properties: *L-smoothness*, *strong convexity*, *bounded variance*, and *uniformly bounded gradient*. It was shown previously that the methods which satisfy these conditions converge to the global optimum solution of the learning task at a rate of $O(\frac{1}{T})$ [14]. Such claims cannot be made for CNNs due to non-convexity and non-linearity.

4 Experimental Analysis

We demonstrate through systematic experiments the performance of FHDnn under various settings. We briefly discuss the datasets and setup for evaluating FHDnn before presenting results for FHDnn for various data distributions for reliable communication. We also compare the resource usage of FHDnn against CNNs. Finally we show evaluation for various unreliable network scenarios.

4.1 Dataset and Platforms

We evaluate FHDnn on 3 different real world datasets: MNIST[7], FashionMNIST[21], CIFAR10 [13]. For performance evaluation we test FHDnn on Raspberry Pi Model 3b and NVIDIA Jetson mobile GPU. We use python and PyTorch for implementing all models. For MNIST, we use a simple network consisting of 2 convolution layers and 2 fully connected layers. For CIFAR10 and FashionMNIST we use the ResNet-18 model [8].

4.2 Accuracy and Compute

We first tune the hyperparameters for both FHDnn and CNNs and analyze their impact by experimenting with E, B, C where E is the number of local epochs, B the local batch size and C the fraction of clients participating in each round. For all experiments we use 100 clients and 100 rounds of communication in order to keep our experiments tractable. We select the best parameters for ResNet and use the same for FHDnn for all experiments in order to allow for a direct comparison.

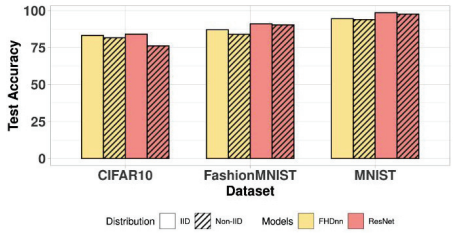

Figure 7: Accuracy of FHDnn and ResNet on different datasets

Table 1: Performance 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

4.2.1 Accuracy. Figure 7 compares the accuracy of FHDnn with ResNet on 3 different datasets for 100 rounds of communication. We observe that for the same number of rounds, FHDnn achieves almost the same accuracy as ResNet and converges faster. Figure 6 shows the smoothed conditional mean across all different hyperparameters for both the models for iid and non-iid distributions. FHDnn reaches an accuracy of 82% in less than 25 rounds of communication whereas ResNet takes 75 rounds for both iid and non-iid data distributions. Moreover the hyperparameters do not have a big influence for FHDnn as seen by the narrow spread (gray region) in Figure 6. Note that the local batch size B doesn't impact FHDnn due to the nature of its training methodology. This allows us to use higher batch sizes up to the constraints of the device, allowing for faster processing whereas B affects the convergence of CNNs.

4.2.2 Compute Resources. The most computationally expensive operation on the client is training. CNN training involves backpropagation for each round which is very expensive. HD on the other hand is very lightweight as featured in Table 1 which quantitatively compares the computation time of FHDnn and ResNet on 2 edge devices for client training. FHDnn is 35% faster and energy efficient than ResNet on RPi and 80% faster and energy efficient on the Jetson mobile GPU.

4.3 Unreliable Communication

In this section we analyze the performance of FHDnn and ResNet under unreliable network conditions as described in Section 3.5. Figure 8 shows the performance of models under each of these network conditions. In order to maintain a direct comparison between CNN and FHDnn we use the same hyperparameters for both models and all experiments. We use $E = 2, C = 0.2, B = 10$ and evaluate the performance on the CIFAR10 dataset. From our experiments we observe that even with fewer clients $C = 0.1$ and for other datasets, the performance of FHDnn is better than ResNet. Due to page constraints we present only the results for the settings mentioned earlier.

4.3.1 Packet Loss. When the packet loss rate is extremely small, below $1e^{-2}$, ResNet has very minimal accuracy loss. However for more realistic packet loss rates such as 20% the CNN model fails to converge. A 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

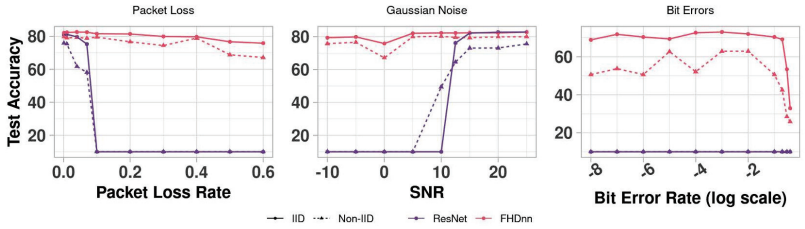

Figure 8: Accuracy comparison of FHDnn with ResNet under unreliable network conditions

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.

4.3.2 Gaussian Noise. We experiment with different Signal-to-Noise Ratios (SNR) to simulate noisy links. Even for lower SNRs like 25dB the accuracy of ResNet drops by 8%. However it's more likely that IoT networks operating on low power wireless networks will incur higher SNRs. For such scenarios FHDnn outperforms ResNet as the latter fails to perform better than random. The accuracy of FHDnn only reduces by 3% which is negligible compared to ResNet.

4.3.3 Bit Errors. Figure 8 shows that CNNs achieve the equivalent of random accuracy even for small bit errors. Since the weights of CNNs are floating point, a single bit flip can significantly change the value of the weights. This compounded with federated averaging hinders convergence. We observe FHDnn incurs an accuracy loss as well, achieving 72% for iid and 69% for non-iid. FHDnn uses an integer representation which is again susceptible to large changes from bit errors. However, our scaling method described in Section 3.5.2 assuages the error to some extent.

4.4 Communication Efficiency

So far we have benchmarked the accuracy of FHDnn for various network conditions. In this section we demonstrate the communication efficiency of FHDnn compared to ResNet.

We compare the amount of data transmitted for federated learning to reach a target accuracy of 80%. We calculate the amount of data transmitted by one client using the formula $data_{transmitted} = n_{rounds} \times update_{size}$, where n_{rounds} is the number of rounds required for convergence by each model. The update size for ResNet with 11M parameters is 22MB while that of FHDnn is 1MB making it 22× smaller. From Section 4.2.1 we know that FHDnn converges 3× faster than ResNet bringing its total communication cost to 25MB. ResNet on the other hand uses up 1.65GB of data to reach the target accuracy.

In Figure 6, we illustrated that FHDnn can converge to the optimal accuracy in much fewer communication rounds. However, this improvement is even more in terms of the actual *clock time* of training. We assume that federated learning takes places over LTE networks where SNR is 5dB for the wireless channel. Each client occupies 1 LTE frame of 5MHz bandwidth and duration 10ms in a time division duplexing manner. For error-free communication, the traditional FL system using ResNet can support up to 1.6 Mbits/sec data rate, whereas we admit errors and communicate at a rate of 5.0 Mbits/sec. Under this setting and for the same experiment as in Section 4.2, FHDnn converges in 1.1 hours for CIFAR IID and 3.3 hours for CIFAR Non-IID on average. On the other hand, ResNet converges in 374.3 hours for both CIFAR IID and CIFAR Non-IID on average.

5 Conclusion

In this work, we presented FHDnn a federated learning framework combining Deep Learning and Hyperdimensional Computing to improve the communication efficiency and reduce the computation costs on edge devices. We detail the architecture, training methodology and evaluate FHDnn through numerous experiments in both reliable and unreliable communication settings and compare its performance with standard CNN. Our experiments show that FHDnn is 66× more communication efficient and lowers client computation costs by 6× while being robust to network errors.

Acknowledgements

This work was supported in part by CRISP, one of six centers in JUMP (an SRC program sponsored by DARPA), SRC Global Research Collaboration (GRC) grant, and NSF grants #1911095, #1826967, #2100237, and #2112167.

References

[1] [n. d.]. UCI machine learning repository. http://archive.ics.uci.edu/ml/datasets/ISOLET.
[2] Ganesh Ananthanarayanan et al. 2017. Real-time video analytics: The killer app for edge computing. *computer* 50, 10 (2017), 58–67.
[3] Austin P Arechiga et al. 2018. The robustness of modern deep learning architectures against single event upset errors. In *2018 IEEE High Performance extreme Computing Conference (HPEC)*. IEEE, 1–6.
[4] Nader Bouacida et al. 2021. Adaptive Federated Dropout: Improving Communication Efficiency and Generalization for Federated Learning. In *INFOCOM WKSHPS*. IEEE, 1–6.
[5] Sebastian Caldas et al. 2019. Expanding the Reach of Federated Learning by Reducing Client Resource Requirements. *arXiv:1812.07210* (Jan 2019).
[6] Ting Chen et al. 2020. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*. PMLR, 1597–1607.
[7] Li Deng. 2012. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine* 29, 6 (2012), 141–142.
[8] Kaiming He et al. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.
[9] Pan Hu et al. 2020. *Starfish: Resilient Image Compression for AIoT Cameras.* ACM, New York, NY, USA, 395–408.
[10] 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*.
[11] Pentti Kanerva. 2009. Hyperdimensional Computing: An Introduction to Computing in Distributed Representation with High-Dimensional Random Vectors. *Cognitive Computation* 1, 2 (Jun 2009), 139–159.
[12] Pentti Kanerva. 2009. Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors. *Cognitive computation* 1, 2 (2009), 139–159.
[13] Alex Krizhevsky, Geoffrey Hinton, et al. 2009. Learning multiple layers of features from tiny images. (2009).
[14] Xiang Li et al. 2019. On the convergence of fedavg on non-iid data. *arXiv preprint arXiv:1907.02189* (2019).
[15] Franz Loewenherz et al. 2017. Video analytics towards vision zero. *Institute of Transportation Engineers. ITE Journal* 87, 3 (2017), 25.
[16] Paul J. Marcelis et al. 2017. DaRe: Data Recovery through Application Layer Coding for LoRaWAN. In *IoTDI*. 97–108.
[17] H Brendan McMahan et al. 2016. Communication-Efficient Learning of Deep Networks from Decentralized Data. (2016), 10.
[18] Shadi A Noghabi et al. 2020. The emerging landscape of edge computing. *GetMobile: Mobile Computing and Communications* 23, 4 (2020), 11–20.
[19] Juha Petäjäjärvi et al. 2016. Evaluation of LoRa LPWAN technology for remote health and wellbeing monitoring. In *ISMICT*.
[20] Shuai Tong et al. 2020. Combating packet collisions using non-stationary signal scaling in LPWANs. In *Proceedings of the 18th International Conference on Mobile Systems, Applications, and Services*. 234–246.
[21] Han Xiao et al. 2017. *Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms.* arXiv:cs.LG/1708.07747 [cs.LG]