HCFL: A High Compression Approach for Communication-Efficient Federated Learning in Very Large Scale IoT Networks

Minh-Duong Nguyen, Sang-Min Lee, Quoc-Viet Pham, *Member, IEEE*, Dinh Thai Hoang, *Member, IEEE*, Diep N. Nguyen, *Senior Member, IEEE*, and Won-Joo Hwang, *Senior Member, IEEE*

***Abstract*—Federated learning (FL) is a new artificial intelligence concept that enables Internet-of-Things (IoT) devices to learn a collaborative model without sending the raw data to centralized nodes for processing. Despite numerous advantages, low computing resources at IoT devices and high communication costs for exchanging model parameters make applications of FL in massive IoT networks very limited. In this work, we develop a novel compression scheme for FL, called *high-compression federated learning (HCFL)*, for very large scale IoT networks. HCFL can reduce the data load for FL processes without changing their structure and hyperparameters. In this way, we not only can significantly reduce communication costs, but also make intensive learning processes more adaptable on low-computing resource IoT devices. Furthermore, we investigate a relationship between the number of IoT devices and the convergence level of the FL model and thereby better assess the quality of the FL process. We demonstrate our HCFL scheme in both simulations and mathematical analyses. Our proposed theoretical research can be used as a minimum level of satisfaction, proving that the FL process can achieve good performance when a determined configuration is met. Therefore, we show that HCFL is applicable in any FL-integrated networks with numerous IoT devices.**

***Index Terms*—Communication efficiency, deep learning, distributed learning, federated learning, autoencoder, data compression, Internet-of-Things, machine type communication.**

I. Introduction

The rapid increase in Internet-of-Things (IoT) applications has revolutionized the big data and machine learning (ML) technology fields [1]. Conventionally, ML and big data algorithms are deployed in the centralized servers [2]. This central deployment can be considered as an effective approach if all the data are available in one consolidated database. Therefore, the whole big data distribution can be processed without any problems such as non independent and identically distributed (non-IID) and lack of sampling population, which is unable to represent the whole data distribution. The arrival of the golden era of massive IoT, in addition to the development of the state-of-the-art 5G and future 6G wireless systems, has expedited the demands for distributed learning solutions at the edge of the network [3], [4].

FL [5], an emerging AI paradigm, has recently attracted considerable attention from the research communities thanks to its nice features and wide-ranging applications. In conventional ML techniques, a server collects the user data and performs a computing process centrally, thereby increasing the risk of high communication overload and data leakage. FL, on the other hand, aims to mitigate the privacy concerns by locally training the ML models, and thus the users only need to dispatch to the server their model parameters [6]. Therefore, this paradigm preserves the private and collaborative aspects of the ML framework for distributed users. As a result, a number of AI applications leveraging the FL process for IoT networks have been developed recently, such as vehicular cloud [7], remote applications [8], smart healthcare [9], and mobile edge networks [10].

Despite recent advancements in computing hardware and computing paradigms such as fog computing and mobile edge computing (MEC), the limitation of communication resources still remains as an obstacle in IoT systems [11]. With the fact that the number of IoT devices is increasing dramatically in the modern world and the wireless resources are limited, communication proficiency sets off to be one of the key challenges for carrying out massive IoT scenarios in which the large-scale FL system is integrated. Therefore, some approaches to address the communication efficiency problem for FL processes have been introduced recently [12]. For example, the authors in [13] utilize the submodel framework, wherein the clients decide which parts are necessary for the model update process instead of implementing the whole model. The FedBoost algorithm in [14] is a combination of an ensemble learning and the random sampling theory. To be more specific, FedBoost selects a random partial set of clients to deliver the updates instead of sending all parameters to every client in each round. The secure aggregation approach proposed in [15] and [16] utilizes the random sampling method to cross-validate the model updates between different pairs of clients before forwarding the model parameters to the aggregation module at the server. Furthermore, the authors in [17] and [18] formulate the optimization problems to minimize the time

Minh-Duong Nguyen and Sang-Min Lee are with the Department of Information Convergence Engineering, Pusan National University, Busan 46241, Republic of Korea (email: {duongnm@pusan.ac.kr, pms88520@pusan.ac.kr}).

Quoc-Viet Pham is with the Korean Southeast Center for the 4th Industrial Revolution Leader Education, Pusan National University, Busan 46241, Republic of Korea (e-mail: vietpq@pusan.ac.kr).

Dinh Thai Hoang, and Diep N. Nguyen are with the School of Electrical and Data Engineering, University of Technology Sydney, Sydney, NSW 2007, Australia (e-mail: {hoang.dinh, diep.nguyen}@uts.edu.au).

Won-Joo Hwang is with the Department of Biomedical Convergence Engineering, Pusan National University, Yangsan 50612, Republic of Korea (e-mail: wjhwang@pusan.ac.kr).

This work was supported by a National Research Foundation of Korea (NRF) Grant funded by the Korean Government (MSIT) under Grants NRF-2019R1C1C1006143 and NRF-2019R1I1A3A01060518.

cost for each communication round [17] and maximize the aggregation data rate from FL users [18]. In order to reduce the communication load during the propagation phase, many researches have applied sparsification and model pruning into FL. Authors in [19] inherit the simple regularization method to develop pruneFL algorithm. This algorithm can reduce the number of parameters up to 7 times. PruneFL, on the other hand, requires more than 6000 communication rounds to reach convergence, which is inappropriate for IoT networks. CA-DSDG [20] and CE-FedAvg [21] both employ the same sparsification concept, in which model parameters that do not change much are dropped from the updating phases. However, the efficiency by sparsification is not convincing where the achieved compression ratio is capped at 70%. Another effective method which is both efficient for communication and computation is model quantization. The works in [22]–[24] focus on model compression wherein the entire network structure is estimated and incorporated into a new model with a low definition model parameter set. The full-precision weights are estimated into approximate set with ternary format (which includes only three values: -1, 0, 1) [25]. Since the deep network parameters are quantized into 2-bit values, the method already reaches its compression limitation. Therefore, the maximum compression efficiency that this method can achieve is constrained by 90%.

In order to further improve the communication efficiency of FL processes, in this work, we propose a new compression strategy, called *high-compression federated learning (HCFL)*, for IoT networks with a very large number of IoT devices. Remarkably, thanks to the compact design on the presentation layer of the OSI model [26], HCFL is strictly in compliance with the entity-specific information flow between current 3GPP interfaces while being communication efficient. Thus, our proposed HCFL scheme can be cooperated with other communication-efficient methods to improve the performance of the whole FL process. We can sum up our contributions in this paper as follows:

- We introduce a novel compression scheme, called *HCFL*, for FL. We leverage an under-complete autoencoder structure in the FL process to develop a communication-efficient strategy that can reduce the transmission load of the FL mega-system with a very large number of IoT devices.
- We provide theoretical analysis of the proposed HCFL method under the FL mega-system. We demonstrate that the HCFL method can achieve a very high performance, e.g., the models with HCFL can be compressed more than 95% of their original size. Furthermore, we investigate a relationship between the system performance and the other compress-aided FL process's characteristics such as the number of IoT devices and the compressor reconstruction error.
- We conduct various simulations with an in-depth comparison with the conventional FL process in terms of performance and communication efficiency. We evaluate the proposed HCFL method under different conditional simulations, including compression efficiency at different compression ratios, convergence analysis at distinct FL process and client predictor settings. The simulation results reveal that the HCFL method can achieve a compression ratio up to 32 times better than that of the conventional FL process. Notably, the trade-off between compression ratio and performance is acceptable when the accuracy loss on the HCFL system is less than 3% in comparison with the baseline method.

The rest of this article is organized as follows. Section II briefly reviews the fundamental knowledge about FL and autoencoder concept. Section III demonstrates the theoretical analysis of HCFL system and how the HCFL and FL process support each other. Section IV details the deployment strategy of the compression-aided federated system along with two appropriate compression model structures. Simulations conducted to compare the performances of a standard FL process and the proposed method in terms of global loss, data reconstruction error and convergence rate are described in Section V. Finally, conclusions are presented in Section VI.

II. Background and Methods

This section first presents the system model, then provides some preliminaries of the conventional FL procedure and its essential formulations, and finally shows the main features and definitions of the autoencoder model.

A. System Model

We consider a mega FL-integrated IoT system involving one server and a set of K clients (i.e., IoT devices), as illustrated in Fig. 1. Clients and server jointly build a shared ML model for processing and data analysis. Each client collects data from its sensors or users. We assume that all the clients' datasets have the IID property in this study [5]. The client's data is processed through an ML-based predictor on each device distributively. Because a large number of clients participate in each communication round, every participating client needs to share limited bandwidth resources. Thus, the transmission rate of each client is also limited. This problem leads to the need for communication efficiency in FL-integrated IoT networks and motivates us to complete this work. Our work aims to reduce the communication burden significantly in the meta FL-integrated IoT system.

B. Federated Learning

FL is an up-to-the-minute concept of ML. In FL, instead of collecting the data and utilizing them for central training at one location, the data is processed distributedly at the clients. The cooperation between the server and clients is to ensure that the model losses are verified at the server, and the improvement is updated onto the distributed clients gradually, as shown in Fig. 1. The FederatedAveraging (FedAvg) algorithm, proposed in [5], utilizes a sum averaging function to combine loss and weight values from different clients as follows:

$$f(w) = \frac{n_k}{n}\sum_{k=1}^{K} F_k(w) \quad \text{where} \quad F_k(w) = \frac{1}{n_k}\sum_{i \in P_k} f_i(w), \tag{1}$$

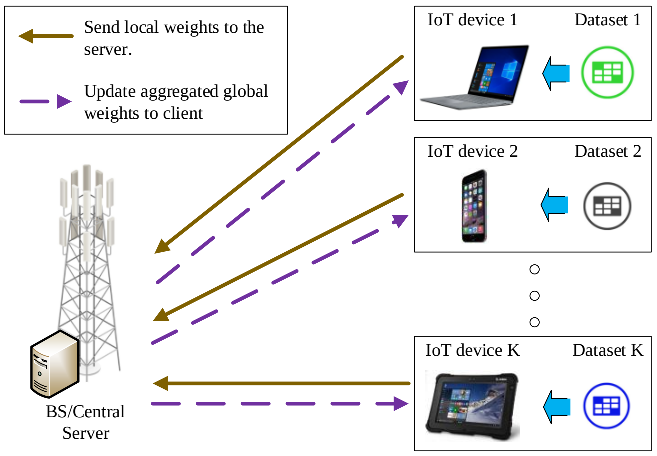


Figure 1. An illustration of FL. The users train the local predictors on their distributed devices, and then the trained local neural networks are sent to the server. The server then executes the aggregation to generate the global model parameters. Lastly, the latest global model is uploaded back to the users, and the process continues until the global model converges.

Here, w denotes the parameters of the FL model, $F_k(w)$ and $f_i(w)$ denote the loss value on the client k and each data point of the whole dataset with a size n, respectively. The dataset is distributed over K clients with a data size of n_k for each client in FL. The gradient on the client k is given as $g_k = \delta F_k(w_k)$. The aggregation function at the server generates a close-form function with the loss sum-averaging update as follows:

$$w_{t+1} \leftarrow \sum_{k=1}^{K} \frac{n_k}{n} w_{t+1}^{k}. \tag{2}$$

The equality $\frac{n}{n_k} = K$ is valid when the dataset on each client is IID with each other on the whole set of dataset of the system [27] and every client samples the same amount of data each communication round. Thus, the FedAvg algorithm updates the model weights as follows:

$$w_{t+1} \leftarrow \frac{1}{K} \sum_{k=1}^{K} w_{t+1}^{k}. \tag{3}$$

C. Autoencoder

Autoencoders utilized in deep learning [2] are dual-symmetric neural networks designed to produce the output data that is approximately-but-not-equal to the intake of that system. Autoencoders learn a concentrated representation of the input while the most valuable information is retained. There are two components in the aforementioned system: an encoder that processes a transformation: $h = f_\omega(x)$, and a decoder $\hat{x} = g_{\omega'}(h)$ that is responsible for the reconstruction. The f and g represent the encoder and decoder neural networks, respectively, and h denotes the feature vector of the model in the aforementioned functions. The encoder f transforms the incoming matrix into a completely divergent model, whereas the decoder g is tasked with rehabilitation. The parameters of the proposed network is learned simultaneously on the task of reconstructing the original output.

The loss function $\mathcal{L}(w)$, which is a measure of the discrepancy between the input x and the output $\hat{x}$, is applied to obtain

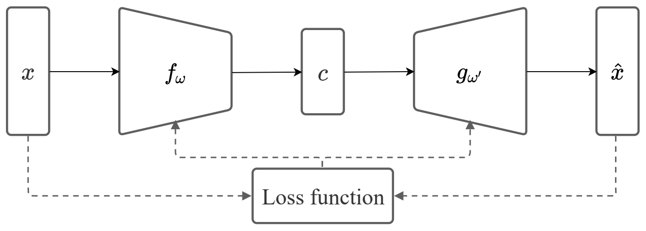


Figure 2. Structure of a general autoencoder.

the lowest possible reconstruction error [28], which is defined as follows:

$$\mathcal{L}(w) = \frac{1}{2}\|\hat{x} - x\|_2^2 = \frac{1}{2}\sum_{k=1}^{N} (\hat{x} - x)^2. \tag{4}$$

The proposed back-propagation operation uses the conventional mean squared error (MSE) loss function [2] to compute the Euclidean distance between the two vectors of the output and input of the autoencoder.

The illustration of a general autoencoder is shown in Fig. 2. Autoencoders are designed to not copy the output from the input perfectly, as aforementioned. Usually, restrictions are conventionally imposed to make the distribution of the output data to be exactly the same with that of the original information. Because of the objective to prioritize the input aspects, which can represent the whole data distribution, the encoder manages to collect the valuable properties only. The main objectives of autoencoders are conventionally feature extraction, dimensionality reduction, and data augmentation. Autoencoders learn the data characterization in a reduced dimensional space by placing additional focus on essential features while attempting to eliminate redundancy and noise. It is based on the encoder-decoder architecture, wherein the encoder encrypts the data with a high dimensional space to the lower-dimensional version. Moreover, the decoder performs in a reverse way, i.e., the decoder reconstructs the initial high-dimensional information from the reduced-dimensional data.

III. HCFL: A High Compression Federated Learning Approach

In this section, we theoretically analyze the convergence of HCFL-integrated FL when applied in IoT networks which consist of a large number of devices. We first evaluate the performance of FL by considering its impacts on HCFL. We then prove that in an FL process with a lot of clients in general or in FL-integrated massive IoT network system in particular, HCFL can achieve a performance which is as good as that of the conventional FL process. Furthermore, we analyze the performance of HCFL model and provide more detailed analysis on relationship between the model reconstruction error and three features, including model complexity, compression ratio, and original data entropy.

A. HCFL: Overview and Convergence Analysis

Using the FedAvg method [5], we can demonstrate the convergence properties of FL when implementing our proposed

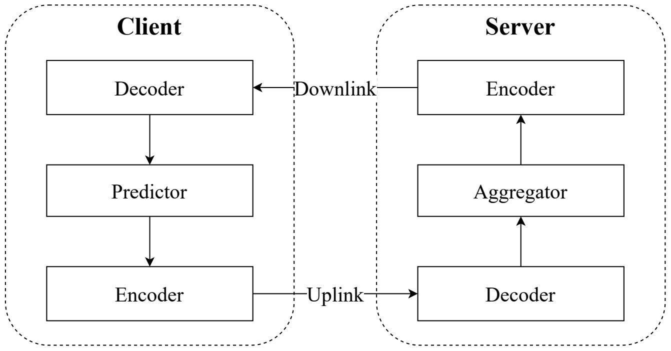


Figure 3. Structure of a general autoencoder with HCFL compression.

HCFL scheme. As the number of clients (i.e., IoT devices) increases, it is guaranteed that aggregated weights are close to the initial desired values.

We consider a compression-integrated FL scenario between a single client in a group of selected clients and a server as shown in Fig. 3. The proposed compression consists of two components: the encoder (which is integrated in the clients) and the decoder which extracts the compressed model sent by the client through the propagation channel. By applying the undercomplete autoencoder [2], we can compress the original data by reducing its dimensions and important information simultaneously. The ratio of input data to the output data of the encoder, which is equal to the ratio of the output data to the input data of the decoder, represents the compression ratio of the model compressor.

The autoencoder only extracts useful information on the distribution of the data due to the reduced dimension of the encoded data. The number of faded features increases as we configure the HCFL at a higher compression ratio [2] (the HCFL compression ratio is defined by the proportion between the encoder input size and the encoder output size). Thus, the decrypted data at the output of the decoder in the server always have a reconstruction error rate in comparison with the original data. The problem of autoencoder error and the capability of the compression in FL is simplified by converting original form of the autoencoder's representation into another close-form. Taken account of probability, the neural network is demonstrated in a synthesized model with many layers of hidden causal variables [29].

$$P(x, g^1, \ldots, g^l) = P(x|g^1) \ldots P(g^{l-2}|g^{l-1}) P(g^{l-1}, g^l), \tag{5}$$

where all the conditional layers $P(g^l|g^{l+1})$ are factorized distributions which are easy for computation of probability and sampling. Alternatively, each of the hidden layer can be represented by:

$$P(g^l|g^{l+1}) = \prod_{j=1}^{n} P(g^l_j|g^{l+1}). \tag{6}$$

The model of the autoencoder consequently can easily be described through a Markov Chain with a close form of a system model similar to that of the Kalman filter, which can be built on linear operators that are perturbed by errors including Gaussian noise [30]. It may be assumed that the multi-layer autoencoder model has a series of $\hat{x}_i$ as output values in discrete time space, where i is the index of the discrete-time event. $\hat{x}_i$ is the total of the original data input x_i with Gaussian noise v_i, which is IID and drawn from a zero-mean distribution as follows:

$$\hat{x}_i = x_i + v_i. \tag{7}$$

The aggregation model proposed in (3) was used to aggregate the decoded weights at the server according to the following equation:

$$\begin{aligned} w_t &= \frac{1}{K} \sum_{k=1}^{K} (w_t^k + v_t^k) \\ &= \frac{1}{K} \sum_{k=1}^{K} w_t^k + \frac{1}{K} \sum_{k=1}^{K} v_t^k. \end{aligned} \tag{8}$$

A set of noise data at weight t is given as: $V_t = \{v_t^1, \ldots, v_t^K\}$. Assume $S_K = \sum_{k=1}^{K} v_t^k$ the following equation is obtained:

$$E\left(\frac{S_K}{K}\right) = \mu. \tag{9}$$

μ is the expected value of the distribution V, and $E(\cdot)$ denotes the expectation function in (9). Furthermore, V_t is regarded as a sequence of serially uncorrelated random variables because of their white noise property. Therefore, we have $\text{Cov}(v_t^i, v_t^j) = 0$ $(\forall i, j)$, where $\text{Cov}(v_t^i, v_t^j)$ is the covariance of two noise values corresponding to user i, j, and sampled from the set of noise data V_t. Hence, we have the following function:

$$\begin{aligned} E\left(\frac{S_K}{K} - \mu\right)^2 &= E\left(\frac{{S_K}^2}{K^2}\right) - E\left(\frac{S_K}{K}\right)^2 \\ &= \frac{1}{K^2}\left[E\left({S_K}^2\right) - E\left(S_K\right)^2\right] \\ &= \frac{1}{K^2}\left(\text{var}(S_K)\right) \\ &= \frac{1}{K^2}\left[\sum_{i=1}^{K} \text{var}(v_t^i) + \sum_{i,j=1}^{K} \text{Cov}(v_t^i, v_t^j)\right] \\ &= \frac{1}{K^2}\left(\text{var}(v_t^1) + \ldots + (\text{var}(v_t^K)\right), \end{aligned} \tag{10}$$

and the variance of each v_t^k can be formulated as

$$\text{var}(v_t^k) = \frac{1}{K} \sum_{k=1}^{K} (v_t^k)^2 - \left(\frac{1}{K} \sum_{k=1}^{K} (v_t^k)\right)^2. \tag{11}$$

Applying the functions (4) and (7) into (11) yields:

$$\begin{aligned} \text{var}(v_t^k) &= \frac{1}{K} \sum_{k=1}^{K} (v_t^k)^2 - \left(\frac{1}{K} \sum_{k=1}^{K} (v_t^k)\right)^2 \\ &\leq \frac{1}{K} \sum_{k=1}^{K} (v_t^k)^2 = \frac{2}{K} \mathcal{L}(w). \end{aligned} \tag{12}$$

From (10) and (12), we have

$$E\left(\frac{S_K}{K} - \mu\right)^2 \leq \frac{2}{K^2} \mathcal{L}(w). \tag{13}$$

It can be observed that, the function (13) gets converged at zero when the number of FL users K approaches infinity. It is evident that $\frac{S_K}{K}$ also converges at μ. As v_i is drawn from a standard normal distribution, S_K is also drawn from the same distribution and has the mean value $\mu = 0$. We have:

$$\lim_{K\to\infty} \frac{S_n}{K} = \lim_{K\to\infty} \frac{1}{K}\sum_{k=1}^{K} v_t^k = 0. \tag{14}$$

Thus, when the number of clients in the FL model is large enough, the parameters aggregated at the server (3) converge at their expected values:

$$\begin{aligned} \lim_{K\to\infty} w_t &= \lim_{K\to\infty} \frac{1}{K}\sum_{k=1}^{K} \hat{w}_t^k \\ &= \lim_{K\to\infty} \left(\frac{1}{K}\sum_{k=1}^{K} w_t^k + \frac{1}{K}\sum_{k=1}^{K} v_t^k\right) \\ &= \lim_{K\to\infty} \frac{1}{K}\sum_{k=1}^{K} w_t^k. \end{aligned} \tag{15}$$

The following paragraphs discuss the relationship between the number of clients, the loss function of autoencoder and the accuracy rate of the aggregated data at the server. Applying Chebyshev's inequality [17] into (13), we have the following formulation to prove the aggregated model convergence:

$$P\Big(\left|w_t - w_t^k\right| < \alpha\Big) = P\left(\left|\frac{S_K}{K} - \mu\right| < \alpha\right), \tag{16}$$

where α denotes the desired low-threshold of the weight deviation on each client. We can take the aggregated model accuracy as the certainty when $\left|w_t - w_t^k\right|$ is larger than a determined value α in the left side of the function. Applying the inequality in [31] yields

$$P\Big(\left|w_t - w_t^k\right| \geq \alpha\Big) \leq \frac{1}{\alpha^2} E\left(\frac{S_K}{K} - \mu\right)^2. \tag{17}$$

Apply the inequality (12), we have an upper boundary for the certainty when the deviation of each parameter is larger than a predefined threshold:

$$P\Big(\left|w_t - w_t^k\right| \geq \alpha\Big) \leq \frac{2}{(K\alpha)^2}\mathcal{L}(w). \tag{18}$$

The proposed inequality (18) is the proof of convergence of implementation of the aggregations algorithm at the server when the number of clients in the federated network is large enough. Despite the high loss value from the autoencoder, we can achieve the aggregated model, which is close to the ideal value of the model weights without the HCFL compression. For instance, the uncertainty of an event when $\mathcal{L}(w) = 2.5$, an expected loss of the aggregated weights compared to the client weights is $\alpha = 0.01$, and $K = 10000$ FL devices, is: $P\Big(\left|w_t - w_t^k\right| \geq \alpha\Big) \leq \frac{2}{(10000*0.01)^2} 2.5 = 0.0005$. Therefore, the certainty is equal to 99.95%. According to [4], the number of IoT and mobile devices with MEC capabilities will be extremely large in beyond 5G and future 6G systems, i.e., 24 billion devices by 2030 [4]. As a result, the certainty induced by the proposed HCFL system can be further reduced emerging scenarios with massive IoT connectivities, showing the efficiency and great potential of the HCFL system.

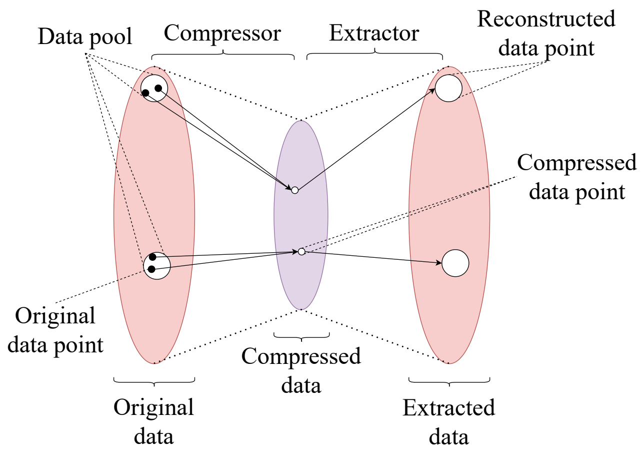


Figure 4. Visualization of mutual information between original data and extracted data in HCFL.

B. HCFL: Compression Analysis

In the HCFL system, the encoder aims to extract the representative features of the original data. There is a direct mapping through the neural network from each individual particle from the primordial model to the compressed output at the encoder. The compressed data is then mapped straightforwardly to the extracted output at the decoder. The procedure can be described in Fig. 4. As we can see from the figure, there is a representative vector for each pool of original data points that have the same characteristics. The reconstructed vector is sampled from the previously mentioned pool and can represent a group of data points in that pool. Hence, the compressed data is the mutual information between the original and the reconstructed data. The lower the compression ratio is, the more information that the compressed data transfers from the original data to the extractor. Therefore, the fewer the number of original data features that one compressed data component needs to represent, the better the HCFL can perform. Denote $H(\cdot)$ to be the entropy of the given dataset and $I(\cdot;\cdot)$ as the mutual information between 2 given sets of data, we have

$$\begin{aligned} H(C) &= I(W;\hat{W}) \\ &= H(W) - H(W|\hat{W}) \\ &= H(W) - H(W - \hat{W}|\hat{W}) \\ &\approx H(W) - H(W - \hat{W}), \end{aligned} \tag{19}$$

where W and $\hat{W}$ are the sets of original model parameters $W = \{w_1, \dots, w_N\}$ and reconstructed model parameters $\hat{W} = \{\hat{w_1}, \dots, \hat{w_N}\}$, respectively. $C = \{c_1, \dots, c_M\}$ is the set of compressed data with N and M being the sizes of the model parameters and compressed data, respectively.

Because the HCFL consists of two sequential neural networks, the proposed model can be demonstrated in a Deep Belief Nets [29] structure where each neural node in the network contributes an independent probability to the output of the network's distribution:

$$P(\hat{W}, W) = \prod_{i=1}^{l} P\left(g^i | g^{i+1}\right) = \prod_{i=1}^{l} \prod_{j=1}^{n_i} P\left(g_j^i | g^{i+1}\right), \quad (20)$$

where i and j are the layer index of the neural network and the neural node index on each layer, respectively. The l and n_l denote the total layers and the number of nodes on each layer of the HCFL model, respectively. From (20), we can assume that the noise generated during the HCFL process follows the Gaussian distribution $\mathcal{N}\left(0, E\left[(W - \hat{W})^2\right]\right)$ (as described in III-A). The $E\left[(W - \hat{W})^2\right]$ is the variance of the reconstructed parameter set $\hat{W}$. Therefore, the entropy of the loss between W and $\hat{W}$ yields:

$$\begin{aligned} H(W - \hat{W}) &= H\left(\mathcal{N}\left(0, E\left[(W - \hat{W})^2\right]\right)\right) \\ &= \frac{1}{2} \log(2\pi e) E\left[\left(W - \hat{W}\right)^2\right] \\ &= N \log(2\pi e) \mathcal{L}(w), \end{aligned} \quad (21)$$

where $H(\cdot)$ denotes the entropy of the equivalent information. Hence, we can estimate the loss function $\mathcal{L}$ in terms of the relationship between the input data and compressed data. We have

$$\begin{aligned} \mathcal{L}(w) &\approx \frac{H(W) - H(C)}{N \log(2\pi e)} \\ &= \frac{\sum_N P(w) \log P(w) - \sum_M P(c) \log P(c)}{N \log(2\pi e)}, \end{aligned} \quad (22)$$

where the distribution of C follows the joint distribution $P(X, g^1, \ldots, g^{\mathcal{V}})$ with $\mathcal{V}$ being the depth of the compressor's hidden layer. Thus, we have

$$\begin{aligned} P(c) &= \prod_{i=1}^{\mathcal{V}+1} P(g^i | g^{i+1}) \\ &= \prod_{i=1}^{\mathcal{V}} \prod_{j=1}^{n_i} P(g_j^i | g^{i+1}) \prod_{k=1}^{M} P(c_j | g^{\mathcal{V}}) \\ &= P(g^1, \ldots, g^{\mathcal{V}}) \prod_{k=1}^{M} P(c_j | g^{\mathcal{V}}), \end{aligned} \quad (23)$$

where $P(g^1, \ldots, g^{\mathcal{V}})$ is the joint probability of the V hidden layers neural network of the compressor and the factorized conditional distribution of the compressed output data.

The two equations (22) and (23) give us a prerequisite for the convergence of the HCFL compression. To achieve the HCFL compression system with low restoration error, the HCFL network's complexity and compression ratio need to be taken into account. When the HCFL with higher compression ratio and more complicated input data is applied, the neural network model is needed to be the deeper. Moreover, the data segmentation technique is considered as a robust method when dealing with models with a huge size and tremendously complex structure.

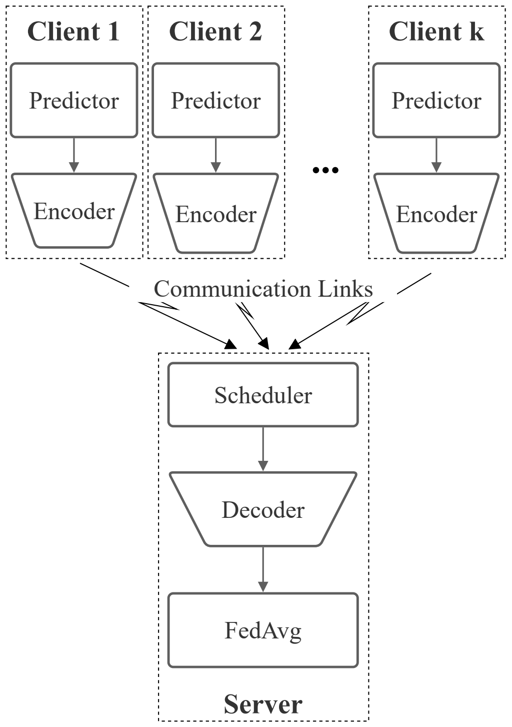


Figure 5. HCFL system deployment.

IV. Proposed Compression-Aided FL Algorithm

In this section, we present how to implement our proposed HCFL-integrated FL in massive IoT networks. We first propose the problem formulation for our HCFL method. Then, we propose the system deployment of HCFL in an FL process and clarify the procedure of the HCFL-aided FL. Secondly, we find that when implementing a traditional multi-layered autoencoder, the performance of the HCFL model is degraded due to the curse of dimensionality. Therefore, we propose new model architectures for the compressor and extractor that improve the integration efficiency of the HCFL into the FL process. Furthermore, we reveal how to pre-process the input model weights and how the model is trained given the model parameter dataset.

A. Problem Formulation

In this part, we aim to develop the framework to minimize the reconstruction error between the original model at input and reconstructed model at output for HCFL in the FL process. In general, HCFL operates as an autoencoder to compress the distributed clients' model and reconstruct the encoded model. Nevertheless, because the clients apply gradient descent stochastically, the models' weight distribution transformation remains undercover. To acknowledge those problems, we need to capture the generalization and transforming trends of every model's distributions, especially when the model's gradient descent is operated in different directions. Thus, our proposed HCFL should be satisfied the two following conditions:

- Minimize the reconstruction error between the original model and reconstructed model as mentioned in (4).
- Maximize the mutual information between the original model and encoded data. Therefore, we can maximize the information transferred from the original data to the encoded data, which improves the performance of the encoder and thus enhance the decoder quality. The optimization problem problem is introduced as follows:

$$\max_{\theta} I(W, C) = D_{\text{KL}}\big[p(W, C) \parallel p(W) \otimes p(C)\big], \quad (24)$$

  where $\otimes$ is the Kronecker product between two matrices W and C, θ is the model parameters of the HCFL model, and the D_{KL} stands for relative entropy between two data distribution.

In this way, we can formulate a joint optimization problem to solve both of the aforementioned tasks. Meanwhile, to conjugate the distance-based problem in (4) and entropic problem in (24), we consider the relationship between MSE and cross entropy (CE). We follow an assumption that the output of deep network in HCFL is demonstrated as Gaussian distribution with variance of σ^2 [2]:

$$\begin{aligned} p(\hat{W} \mid W) &= \mathcal{N}(W, \sigma^2) \\ &= \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2}\frac{(W - \hat{W})^2}{\sigma^2}\right). \end{aligned} \quad (25)$$

Applying CE on the HCFL's output, we have:

$$\begin{aligned} &H(W, \hat{W}) = -E_{p(W)} \log p(\hat{W}) \\ &= -E_{p(W)} \log\Big[\frac{1}{\sqrt{2\pi\sigma^2}} \exp\Big(-\frac{1}{2}\frac{(W - \hat{W})^2}{\sigma^2}\Big)\Big] \\ &= \mathcal{O}\big(-E_{p(W)}(W - \hat{W})^2\big). \end{aligned} \quad (26)$$

The $\mathcal{O}$ describes the growth rate of the function, $E_{p(W)}$ is the expectation over distribution of model parameters W. From the above formula, we can see that $H(W, \hat{W})$ is proportional with MSE loss $E_{p(W)}(W - \hat{W}^2)$. Thus, we have the joint problem formulation as follows:

$$\min_{\theta} \quad L = H - I \quad (27a)$$

$$\text{s.t.} \quad H = \lambda H(W, \hat{W}), \quad (27b)$$

$$I = (1 - \lambda) D_{\text{KL}}\big[p(W, C) \parallel p(W) \otimes p(C)\big], \quad (27c)$$

λ denotes the scale coefficient between two proposed tasks. θ is the parameter set of the HCFL model. The HCFL is thus trained by the following gradient descent process:

$$\theta = \theta - \gamma \nabla L, \quad (28)$$

where γ is the learning rate for the HCFL training process.

B. System Deployment

This part discusses the system deployment of the HCFL in the FL. As we can see from Fig. 5, the HCFL comprises two components: encoders which are located on each client's domain, and a single decoder which is embedded in the server's firmware. Although numerous encoders are required for the activation of the compression operation on the clients, the server's side only requires a single decoder because the incoming client-training-information from all clients is discontinuous and can be scheduled using the First-In-First-Out rule. This ensures that the hardware requirement of the server can still be satisfied.

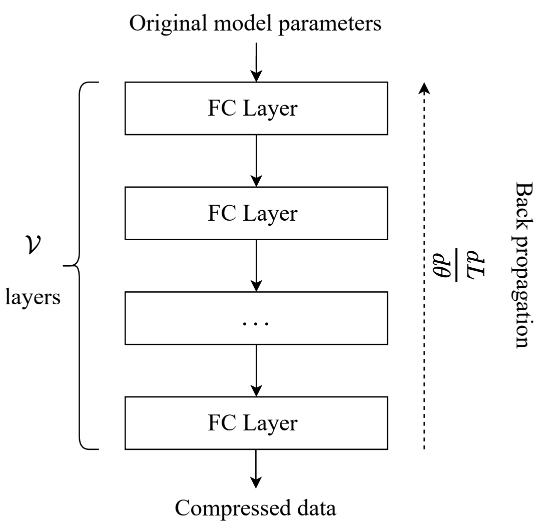
Algorithm 1 shows the complete pseudo-code for the HCFL-integrated FL. After the initialization, the server starts its training iterations. Each iteration is named as one communication round. At every communication round, the server uploads global weights to every client and calls for updates from K selected clients. The selected clients train their models in parallel and then send their new local models to the server. The server decodes all of the received local parameters and then executes the accumulation and averaging to acquire a new global model. This process is continuously processed until the FL achieves the desired convergence or when the max communication round value is reached. Moreover, the encoder compresses the model prior to the packaging and modulating phases of the communication process on the transmitting links. Likewise, the decoder reconstructs the model after the demodulation and de-packaging phases in the receiving links. This closed-loop process ensures the compatibility of HCFL in any IoT systems.

C. HCFL Network Architecture

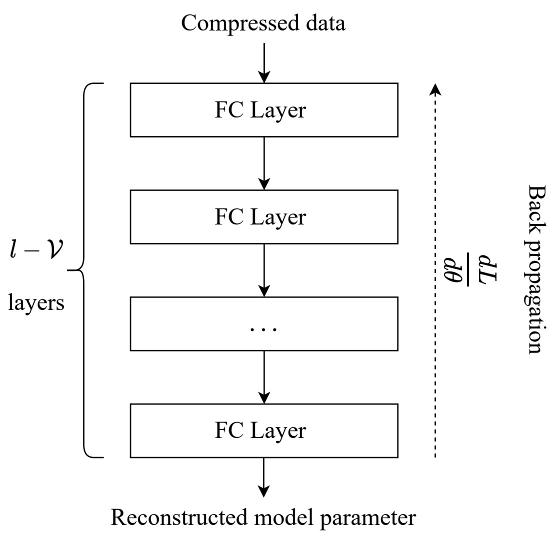
The proposed HCFL framework is expected to compress the model parameter data with a ratio of 1:4 to 1:32 in terms of size. This rate can be achieved by dealing with the loss of data during the encoding scheme. Hence, a deep neural network is leveraged to reduce the loss between the reconstructed and the original data.

1) Definition of Datasets: To develop HCFL, we first define the dataset for the network. The data are extracted from the model parameter dataset. Instead of extracting the model parameters towards the end of the training, the data prepared for this system is generated after each epoch in each client to assist the compressor in learning the values and spatial distributions of the neural network‘s coefficients. The curse of dimensionality and the computation cost at the client's end were reduced by splitting the data into two components to be trained with two different compressors, namely the convolution kernel dataset and dense network dataset, where elements in each part have the similar dispensational characteristics.

2) The Proposed Compression System: uses $\mathcal{V}$ fully-connected (FC) layers at the encryptor (Fig. 6a) and $(l - \mathcal{V})$ FC layers at the extractor (Fig. 6b), where l is the total hidden layers of the HCFL system to activate the dimensionality reduction. Each FC layer consists of a dense layer followed by a Tanh activation function for each layer node. The usage of Tanh is to guarantee the output of the HCFL in the range $[-1, 1]$, which is the value range of original model parameters. The FC layer also uses an additional batch normalization in the input in order to make the HCFL more stable and faster by re-centering and re-scaling, as in Fig. 7. The depth of the HCFL hidden layer is set according to the compression ratio of the HCFL and the complexity of the input model. The higher the HCFL compression ratio is, the more FC layers are added



(a) Architecture of HCFL compressor.



(b) Architecture of HCFL extractor.

Figure 6. HCFL network architecture. There are $\mathcal{V}$ FC layers on the compressor and $l-\mathcal{V}$ FC layers on the extractor, respectively. The FC layers in each HCFL part is connected sequentially.

Algorithm 1 HCFL compression-aided FedAvg FL. We denote k to be the client index in a total number of K clients; E is the local epochs, η is the learning rate, and the local mini-batch size is represented as B.

```
procedure SERVEREXECUTES
    Initialize $w_0$
    for each round $t = 1, 2, \ldots$ do
        $m \leftarrow \max(1, K \times C)$
        $S_t \leftarrow$ (random set of $m$ clients)
        for all $k \in S_t$ in parallel do
            $h^k_{t+1} \leftarrow$ CLIENTUPDATES$(w_t, k)$
            $w^k_{t+1} \leftarrow$ DECODE$(h^k_{t+1})$
            $w_{t+1} \leftarrow \frac{(k-1)}{k} w_{t+1} + \frac{(1)}{k} w^k_{t+1}$
    Updates $w$ to the clients.
procedure CLIENTUPDATES$(w, k)$
    $B \leftarrow$ (divide $P_k$ into batches of size B)
    for each local epoch $e$ from 1 to $E$ do
        for batch $b \in B$ do
            $w \leftarrow w - \eta \nabla l(w; b)$
    $h$ = Encode$(w)$ return $h$ to server
```

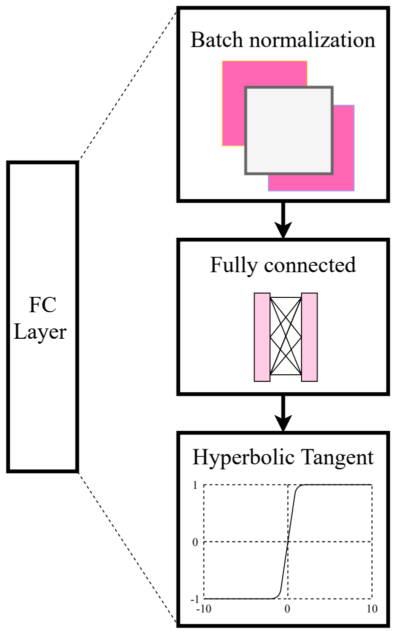


Figure 7. Composition of an FC layer.

into the deep network. As we can see from (22) and (23), as more layers being added to the neural network, the lower bound of the log probability that the model assigns for the training data ascends. Thus, the deep neural networks deliver better compression performance than those of shallow or linear neural networks [32], [33].

3) Data Pre-processing: A detailed description is given below:

- **Data preparation:** The model parameters are stored during the FL execution. To avoid the dataset imbalance, we only fetch the pre-saturated client's predicting models. Moreover, with the proposed method, the HCFL compressor can learn the model general distribution at every learning state.
- **Data segmentation:** We apply the divide-and-conquer algorithm [34] to break down each individual model parameter into two or more sub-datasets whose distributions have the high mutual information. Therefore, the clustered dataset distributions become simple enough that HCFL can avoid the curse of dimensionality.

D. Proposed Training Phase

We adopt the transfer learning technique to implement the

HCFL. Firstly, we train a pre-model with a small amount of dataset on the server. By applying augmentation on small dataset on server, the set of model data samples is thus has an additional variation on data distribution [35]. Thus, we add noise to the gradient, which raises the variance of the norm of the gradient. This high variance thus increases the gradient stochasticity for the pre-train model [36], which improves the model parameters dataset generalization. Therefore, the received dataset is appropriate for the HCFL training phase. Afterward, the model which is obtained at each epoch from the pre-model training phase is utilized for our HCFL model training process. The training process is illustrated in Figures 6a and 6b. In these figures, the HCFL is trained as demonstrated in (28). As explained in Section IV-A, the trained HCFL model thus has the generalized features of the distributed clients' models in practice.

V. Performance Evaluation

In this part, we evaluate the performance of the proposed HCFL compression. Depending on different testing scenarios, we establish multiple settings to examine the HCFL under distinct conditions in Section V-A. We then evaluate the HCFL compression efficiency with various compression ratios in Section V-B. Our experiments show that HCFL can achieve up to 32 times of reduction in data size. Then, we evaluate the impact of clients number on FL convergence rate in Section V-C. Section V-D shows the influence of the client's predictor hyper-parameters on the HCFL-integrated FL to analyze the suitable setting when applying HCFL on FL.

A. Settings

The reconstruction error of the HCFL-compression framework is considered along with the losses of the predictors at the clients ends with the updated model parameters under different FL settings to simplify the evaluation. For the simulation, we use a set of 100 clients in synchronous FL system for assessments. A comprehensive description of the setting is presented below.

Dataset: Because the HCFL only makes transformations on the model weights of the predictors on clients, the HCFL performance only depends on the neural network model regardless of which type of dataset is used. We choose two popular benchmark datasets that are widely used for classification. We focus on the performance of FL which is not affected by the non-IID data, where each client holds a separated dataset from the other.

- *MNIST* [37]: consists of $60,000$ training and $10,000$ testing gray-scale images of ten classes of handwritten digits. Each image has the dimension of 28×28 pixels. For more details, there are 100 clients in the system where each client has 600 samples that are independent and identical with each other. Because of the simplicity of MNIST, this dataset is mostly used to train small networks.
- *EMNIST* [38]: includes $131,600$ images of 47 balanced classes with a size of 28×28 pixels. The dataset is partitioned into 100 independent and identical clusters of $11,280$ images, each is represented for the dataset of each different client.

Table I
Performance of HCFL and other compression methods on LeNet-5 model training with MNIST dataset with different compressing ratio (Communication Cost in 100 rounds on IID Data). Each round, ten out of 100 clients are participated in the training ($C = 0.1$).

Compress Method	Reconstruction error	Encoded Size Up/Download (MB)	True Compress Ratio
FedAvg	0.0	20500/20500	1.000
T-FedAvg	N/A	1281/1281	15.999
HCFL 1:4	0.0016	5170/5170	3.965
HCFL 1:8	0.0037	2620/2620	7.824
HCFL 1:16	0.0037	1370/1370	14.963
HCFL 1:32	**0.0040**	**757/757**	**27.080**

Models: In order to assess the performance of HCFL, we use two deep learning models: LeNet-5 and 5-CNN, which represent two popular convolutional deep network architectures. The configuration is described in detail as follows:

- LeNet-5 [39] is a simple convolutional neural network. It contains the basic units of a convolutional neural network. In LeNet-5, in which the data is first feed-forward through two sets of feature mapping. Each set includes one convolutional layer, with the max-pooling layer is followed subsequently. The two proposed sets are connected sequentially, followed by a fully connected layer that uses ReLU as their main activation function. A ten-node group of softmax is applied at the output layer in order to execute the probability classification for the model.
- 5-CNN is an advanced deep neural network. It comprises two main components: the first component includes five convolutional layers, and the second consists of two fully connected layers. A max-pooling layer is used after each convolutional layer, and a ReLU activation function is applied at the end of each max-pooling layer. Then, additional fully connected layer is applied to process the classification works. Dropout layers are added, followed by each fully connected layer to reduce the impact of overfitting.

Dataset segmentation: The model parameter datasets of the FL model in our simulations are processed as follows:

- *MNIST*: The convolutional layers and dense layers are trained in different HCFL compressors and extractors. Each of the HCFL learns the different distribution of each group of convolution kernel parameters or fully connected weights. Hence, we can achieve the high compression efficiency for the HCFL.
- *EMNIST*: Due to the complexity of the 5-CNN model parameters, we fractionate the dense layers' parameters into 8 balanced parts in order to reduce the entropy of each part. The HCFL compressor needs to execute eight different trainings and the HCFL compressor setting is stored in the HCFL memory. Whenever the FL model

Table II
PERFORMANCE OF HCFL COMPRESSION ON 5-CNN MODEL TRAINING WITH EMNIST DATASET WITH DIFFERENT COMPRESSING RATIO (COMMUNICATION COST IN 100 ROUNDS ON IID DATA). EACH ROUND, 100 OUT OF 100 CLIENTS ARE PARTICIPATED IN THE TRAINING ($C = 0.1$).

Compress Method	Reconstruction error	Encoded Size Up/Download (MB)	True Compress Ratio
FedAvg	0.0	27200/27200	1.000
T-FedAvg	N/A	1,650/1,650	16.485
HCFL 1:4	0.0486	5630/5630	4.831
HCFL 1:8	0.0495	2930/2930	9.283
HCFL 1:16	**0.0501**	**1570/1570**	**17.324**
HCFL 1:32	**0.0693**	**910.55/910.55**	**29.872**

applies the HCFL, the particular HCFL setfting is loaded from the memory to process the corresponding segmented dataset.

Initial implementation details: Each round, the server selects 10 customers from a pool of 100 clients at random. We will use a fixed learning rate of 0.01 as an example. The number of epochs at each client is set to five, and the mini-batch size is set to 64. This implementation is applied to the experiment in Section V-B. In Sections V-C and V-D, the numbers of participating clients K, local epoch E and local batch-size B are applied with different values to evaluate the HCFL under different settings. The detailed settings of those experiments are demonstrated in sections V-C and V-D.

B. *Robustness to Compression Proficiency*

We first perform some experiments with the HCFL compression only to validate the affection of the compression ratio on the reconstruction error of the extracted data. We evaluate the algorithm after 100 training rounds with an equal number of training samples for each client, and every client is computed in 10 independent runs each round for a fair comparison. In this subsection, we consider three main compression efficiency features, including compression efficiency, computational delay, and HCFL-assisted FL accuracy.

From the perspective of compression efficiency, we evaluate the actual compression performance when applying different compression ratio settings in HCFL on FL. Tables I and II show the reconstruction error and the compression efficiency of HCFL and the benchmarks, including FedAvg [5] and T-FedAvg [22], respectively, under different compression ratio settings. We apply the benchmarks on two dataset: MNIST and EMNIST. Assume that there are 10 users participating in each round over 100 rounds, the total capacity which is necessary for the transmission between clients and the centralized server is 20.5 GB for the LeNet-5 model and 27.2 GB for the 5-CNN model, respectively. This large amount of data is an enormous burden on large-scale IoT networks. The communication cost is expected to be more significant in high complexity deep networks (ie., AlexNet [40] or ResNet [41]). The relationship between the model parameter size and the communication time is given as

$$T_k^{comm} = s_k / R_k, \tag{29}$$

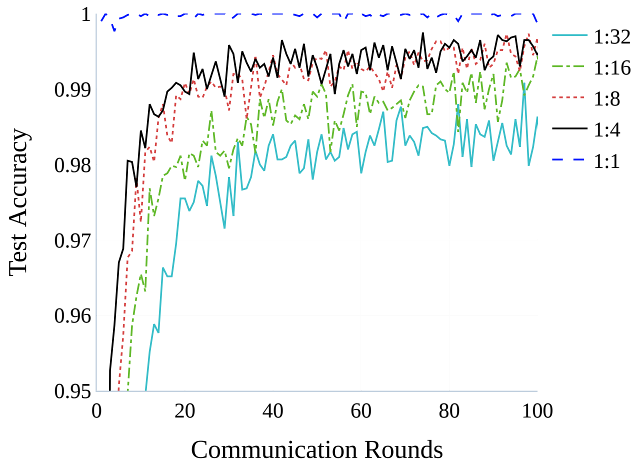


Figure 8. Aggregation accuracy of HCFL on MNIST dataset at different compression ratio settings.

Table III
COMPUTATIONAL DELAY OF HCFL-ASSISTED FL ON LENET-5 MODEL TRAINING WITH MNIST AND ON 5-CNN MODEL TRAINING WITH EMNIST (EVALUATION IS AVERAGED IN 100 ROUNDS OF FL PROCESS). EACH ROUND, TEN OUT OF 100 CLIENTS ARE PARTICIPATED IN THE TRAINING ($C = 0.1$).

Compression Ratio	Computational time (second)			
	LeNet-5-integrated client		5-CNN-integrated client	
	client	server	client	server
Baseline	2.133	0.0142	2.171	0.0228
1:4	2.146	0.1095	2.190	0.2353
1:8	2.178	0.1136	2.192	0.2793
1:16	2.183	0.1143	2.195	0.3130
1:32	2.283	0.1231	2.209	0.3435

where s_k is the model parameter size and R_k is the transmission rate. From (29), by reducing transmitting model parameter size, we can reduce the communication time with the same ratio. It is worth noticing that when we increase the compression ratio, the communication efficiency is improved. In case HCFL-assisted FL is applied, we can reduce a huge amount of data on the transmission link and the required communication time as well. In particular, there are four proposed compression ratios in our work: 1:4, 1:8, 1:16, and 1:32. The data from the two tables reveals that applying the setting of 1:4 helps to reduce the communication cost to 25.22% and 20.7% when working on LeNet-5 model and 5-CNN. Especially, by applying the 1:32 setting, the communication cost can be reduced to 3.7% (from 20.5 GB in the conventional FL to 757 MB in HCFL-assisted FL) in LeNet-5-integrated clients and 3.35% (from 27.2 GB in the conventional FL to 910.55 MB in HCFL-assisted FL) in 5-CNN-integrated clients. Furthermore, it can be observed from Table I that HCFL at compression ratio of 1:32 outperforms the T-FedAvg [22] in communication efficiency as the T-FedAvg's maximum compression ratio is capped at 16 times.

From the perspective of computational delay, we assess the average computational time for both clients and server in a long-term FL process. Because the HCFL components are deployed in both server and clients, we conduct the delay calculations on both centralized server and distributed clients. We calculate the average delay value over 100 communication rounds in order to get the fair assessment. The computation delay is demonstrated in Table III. As observed in the table, the assessment is implemented on both LeNet-5-integrated client (MNIST dataset) and 5-CNN-integrated client (EMNIST dataset). The HCFL process time can be calculated as follows:

$$T_r^{HCFL} = T_r^{comp} - T_1^{comp}, \quad (30)$$

where r is the compression ratio of the HCFL system, T_r^{HCFL} is the HCFL process computation delay and T_r^{comp} is the client's total computation delay with the compression ratio r. From the table, we can see that the HCFL process on both clients and server is low (less than 40 milliseconds on client and 350 milliseconds on server). This time amount is much lower than the predicting process delay on the client (around 2.1 to 2.2 seconds).

In the following evaluation of HCFL compression efficiency, we consider the HCFL-assisted FL accuracy. The assessment uses the same setting with ten clients randomly selected from 100 clients in total, the selected client ratio is set at 0.1, and the batch size is set to the maximum possible value (i.e., equal to the data size at each client) with the number of training epoch set at five. To compare the performance between different compression ratios of HCFL, we train different neural network models for the encoder and decoder of each particular HCFL with the determined compression ratio and then embed them into the recommend FL model. Fig. 8 shows the predicting accuracy at clients with different compression ratio settings at the HCFL compressor. The test accuracy is calculated by the percentage of data in the test set with the predicting label matching their original label. As we can see from the figure, the accuracy of HCFL at the beginning of the distributed training session is relatively low due to the internal error of the data generated by the HCFL compressor when working on MNIST dataset with LeNet-5 as the predicting model. However, the performance of the HCFL-integrated FL can be converged after only six to seven communication rounds. Moreover, although the global accuracy is decreased at the compression rate of 16 and 32, the test accuracy at 98% for 1:32 compression rate is at the acceptable threshold. Alternatively, this decline may be attributed to the fact that the more representation of the data vanishes from the original information, the more error-prone that the HCFL-assisted IoT system is exposed to.

When dealing with such model with higher complexity as 5-CNN for the EMNIST, we need to apply the pre-mentioned dataset segmentation technique for the model parameters. The performance of HCFL on the proposed model is demonstrated in Fig. 9. As observed from the figure, due to the random shuffle and division of the full EMNIST dataset for the clients and the stochastic initiation of the client predicting model, the test accuracy of all five cases shows the high fluctuation and different convergence speeds at the primitive stage of the HCFL training. However, the HCFL system can converge within after less than 100 communication rounds, regardless the proposed compression ratio. The high performance of different compression schemes in the prolonged term is believed to attain on account of the reduction in entropy of the model parameters which is proposed in Section III-B.

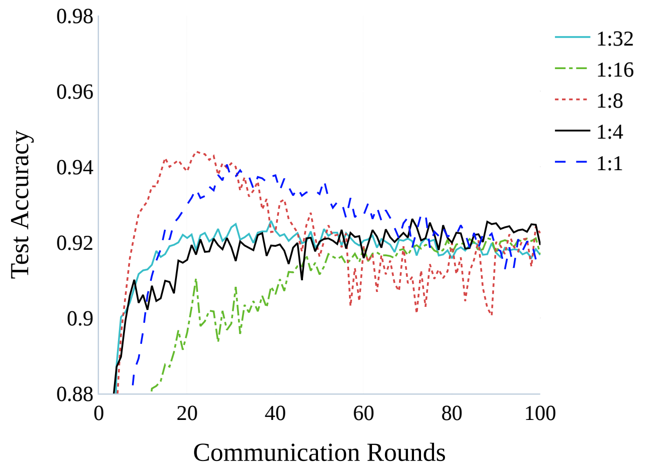
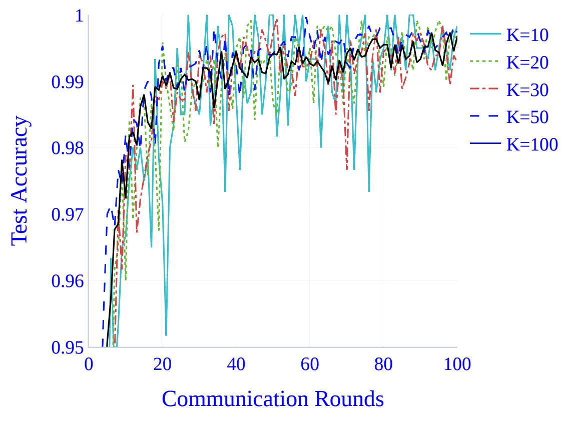


Figure 9. Aggregation accuracy of HCFL on EMNIST dataset at different compression ratio settings.

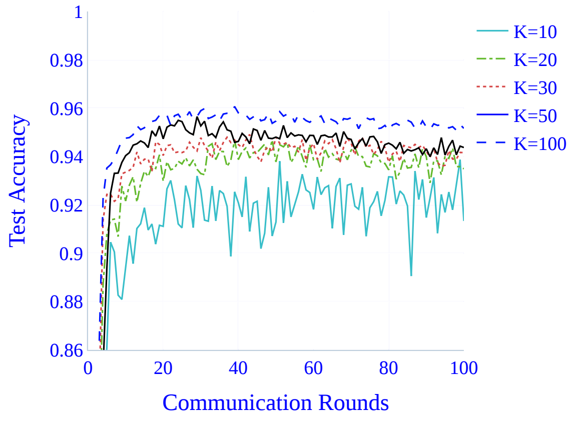
C. *Contribution of Participating Client Quantity to Global Convergence*

In this assessment, we evaluate the impact of the number of participating clients on the convergence of the error-prone compressed data. The HCFL compression is applied in FL with a dissimilar quantity of clients. We evaluate them on two datasets, i.e., MNIST and EMNIST. The results obtained by the HCFL-assisted FL on MNIST and EMNIST are shown in Fig. 10a and Fig. 10b, respectively. In general, it is shown that the performance of the HCFL compression function in the FL model can attain expected efficiency with a various number of clients participating in the FL model. However, the larger the number of clients is, the sooner the system will converge to the global extrema. Therefore, with a large number of clients using HCFL, predictors accuracy can quickly achieve a high performance, and thus the accuracy will be more stable through communication rounds. For example, for the setting $K = 100$, the test accuracy of the LeNet-5 model on MNIST dataset reaches 99% after less than 20 communication rounds with a low standard deviation (less than 1%). In contrast, the system with a low number of assigned clients ($K = 10$) has a relatively high standard deviation of accuracy (more than 3%) after 80 communication rounds.

In particular, different deep network models being used on the clients make distinct impacts on the whole HCFL-assisted FL. For instance, the complex model 5-CNN with the EMNIST dataset shows a more significant outcome than the mentioned evaluation on the MNIST dataset. For the system with 100 clients in comparison with the 10-client system, the test accuracy is higher with a notably lower variance. Generally, in a predetermined number of training rounds, the number of clients affects the FL process accuracy and convergence speed. As more clients participating in the FL

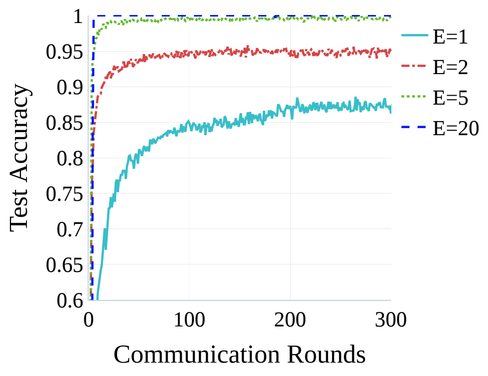


(a)

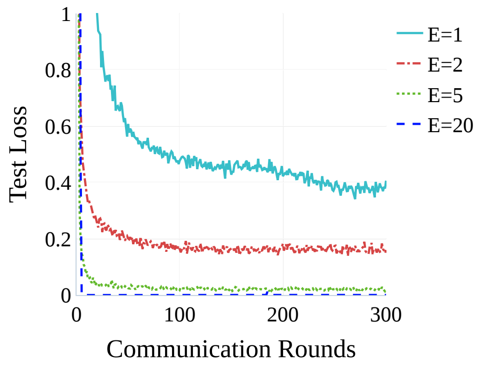


(b)

Figure 10. Implementations of the FL process applying HCFL for MNIST and EMNIST handwritten digit dataset. In these simulations, we set up the FL with different numbers of clients. (a) Number of clients affects the aggregation accuracy of the FL process on MNIST dataset. (b) Number of clients affects the aggregation accuracy of the FL process on EMNIST dataset.



(a)



(b)

Figure 11. Aggregation accuracy on LeNet-5 model with MNIST dataset for different local epochs (E). Number of total clients and participated ratio are fixed ($K = 100, C = 0.1$).

process each round, the model's absolute precision and training speed suffer from fewer adverse effects. Nevertheless, once K is improved to a particular level, the advancement of the system performance will be less noteworthy, and sometimes it even sets about degrading. When bringing the FL process into practice, we can face difficulties that as K expands, more and more clients update their local parameters to the server. As a consequence, the communication and computation cost of the FL model amplify significantly. Fortunately, as we can see in the simulation results in Fig. 10a and Fig. 10b, there always exists an upper bound of performance saturation when K increases. This is encouraging because in real-world applications with large scale of clients, we only need to select a set of clients from the network to execute the FL process in each communication round. This procedure save a considerable amount of communication cost for the FL process.

D. *Contribution of Distributed Model Hyper-parameters to Global Convergence*

In this part, we investigate the impact of compression reconstruction error on different FL process settings. Two client model's hyper-parameters that we consider are: Epoch and batch size. From the perspective of epoch analysis, as the number of epochs training on each selected client at each communication round increases, the system can achieve significantly better test accuracy (Fig. 11a) and loss (Fig. 11b) after each round. To be more specific, the system in which the number of epochs is set at 20 can converge after few rounds of communications. Meanwhile, the system with a minimum setting of training epoch gets saturated at the accuracy of around 86% and the loss of 0.4. The number of epochs affects the number of iterations that the model completes the training throughout the entire training dataset. Hence, the predictor can achieve the better performance when we train the model with more epochs. However, when the number of epochs becomes too large, the system can suffer from overfitting and extreme computational burden. In this work, we use five epochs for a balanced trade-off between the system computation and the system performance since the convergence of that setting is close to the capability of the pre-mentioned high epoch setting.

In the following analysis, the batch size is taken into account. As observed from Fig. 12a and Fig. 12b, HCFL-integrated FL can achieve better performance when the small

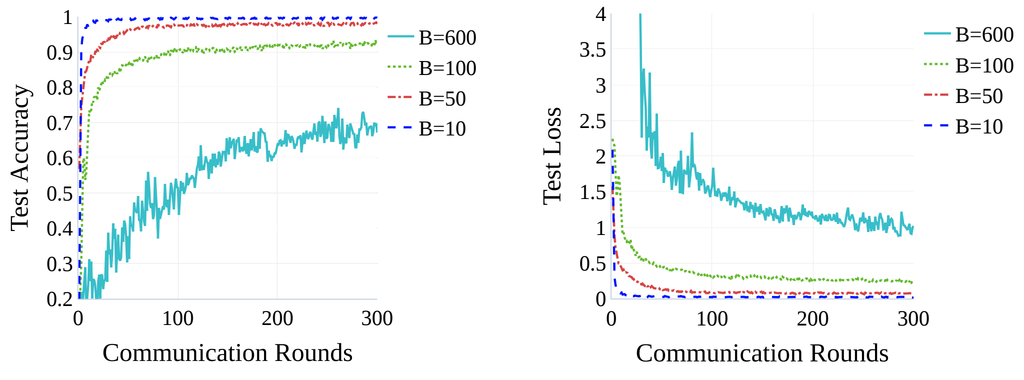


Figure 12. Aggregation accuracy on LeNet-5 model with MNIST dataset for different batch size (B). Number of total clients and participated ratio are fixed ($K = 100, C = 0.1$), number of epoch ($E = 5$).

batch size is applied. The test loss reflects the categorical cross-entropy between the original and predicted labels of the data collected on clients. The maximum batch size can hardly attain the expected efficiency with a low-par accuracy and loss after 300 communication rounds (60% and more than one respectively). It can be observed that, the setting of batch size can make a big contribution to the computation load of every client. Reducing the batch size helps the networks train faster and requires less memory with mini-batches. Moreover, the weight updates have more variance when applying a smaller batch size than when we use the larger batch size. This noise can act as a regularizing effect in weight updating, making the FL operate with better performance.

VI. Conclusion and Future Works

A new compression scheme for FL was developed in this study. In particular, the proposed technique possesses several advantages over the conventional technique, such as high compression proficiency, high accuracy, and viability for the FL process (especially the FedAvg algorithm, wherein it is embedded at the server). For example, when the HCFL-integrated FL was set at the high compression ratio (i.e., 1:32), the global test accuracy of the FL process was reduced by approximately $1\% - 2\%$ in comparison to the baseline method. It was confirmed that the HCFL can be implemented to compensate for the susceptibility to the error of the autoencoder reconstruction output. Further, the low complexity of the proposed compression scheme makes it highly suitable for the FL process, especially the state-of-the-art machine type communication systems such as unmanned aerial vehicle and smart city applications.

In future work, we aim to improve the HCFL proficiency and reduce the supervised-training dependencies by applying deterministic algorithms on the encoder phase. Thus, HCFL is then only needed to train the decoder to adapt to the encoder's behavior. Moreover, multi-task learning is a promising method to aid the autoencoder's performance when implemented on deeper deep models such as ResNet, DenseNet, AlexNet, or VGG-16. Hence, HCFL can further be applied to a wide range of FL concepts.

References

[1] F. Hussain, S. A. Hassan, R. Hussain, and E. Hossain, "Machine learning for resource management in cellular and IoT networks: Potentials, current solutions, and open challenges," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 2, pp. 1251–1275, Jan. 2020.

[2] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press Cambridge, Nov. 2016.

[3] X. Chen, D. W. K. Ng, W. Yu, E. G. Larsson, N. Al-Dhahir, and R. Schober, "Massive access for 5G and beyond," *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 3, pp. 615–637, Mar. 2021.

[4] C. De Alwis, A. Kalla, Q.-V. Pham, P. Kumar, K. Dev, W.-J. Hwang, and M. Liyanage, "Survey on 6G frontiers: Trends, applications, requirements, technologies and future research," *IEEE Open Journal of the Communications Society*, pp. 836–886, Apr. 2021.

[5] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Artificial Intelligence and Statistics*. PMLR, Feb. 2017, pp. 1273–1282.

[6] D. C. Nguyen, M. Ding, Q.-V. Pham, P. N. Pathirana, L. B. Le, A. Seneviratne, J. Li, D. Niyato, and H. V. Poor, "Federated learning meets blockchain in edge computing: Opportunities and challenges," *IEEE Internet of Things Journal*, Apr. 2021, early access.

[7] A. Du, Y. Shen, and L. Tseng, "CarML: Distributed machine learning in vehicular clouds," in *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*, ser. MobiCom '20. London, United Kingdom: Association for Computing Machinery, Sep. 2020.

[8] H. Zhang, A. Zhou, J. Lu, R. Ma, Y. Hu, C. Li, X. Zhang, H. Ma, and X. Chen, "OnRL: Improving mobile video telephony via online reinforcement learning," in *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*, ser. MobiCom '20. London, United Kingdom: Association for Computing Machinery, Sep. 2020.

[9] Y. Chen, X. Qin, J. Wang, C. Yu, and W. Gao, "FedHealth: A federated transfer learning framework for wearable healthcare," *IEEE Intelligent Systems*, vol. 35, no. 4, pp. 83–93, Apr. 2020.

[10] W. Y. B. Lim, N. C. Luong, D. T. Hoang, Y. Jiao, Y.-C. Liang, Q. Yang, D. Niyato, and C. Miao, "Federated learning in mobile edge networks: A comprehensive survey," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 3, pp. 2031–2063, Apr. 2020.

[11] Q.-V. Pham, H. T. Nguyen, Z. Han, and W.-J. Hwang, "Coalitional games for computation offloading in NOMA-enabled multi-access edge computing," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 2, pp. 1982–1993, Feb. 2020.

[12] M. Chen, N. Shlezinger, H. V. Poor, Y. C. Eldar, and S. Cui, "Communication-efficient federated learning," *Proceedings of the National Academy of Sciences*, vol. 118, no. 17, Apr. 2021.

[13] C. Niu, F. Wu, S. Tang, L. Hua, R. Jia, C. Lv, Z. Wu, and G. Chen, "Billion-scale federated learning on mobile clients: A submodel design with tunable privacy," in *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*, ser. MobiCom '20.

London, United Kingdom: Association for Computing Machinery, Sep. 2020.

[14] J. Hamer, M. Mohri, and A. T. Suresh, "FedBoost: A communication-efficient algorithm for federated learning," in *International Conference on Machine Learning*. PMLR, Jul. 2020, pp. 3973–3983.

[15] K. Bonawitz, F. Salehi, J. Konečný, B. McMahan, and M. Gruteser, "Federated learning with autotuned communication-efficient secure aggregation," in *2019 53rd Asilomar Conference on Signals, Systems, and Computers*, Nov. 2019, pp. 1222–1226.

[16] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth, "Practical secure aggregation for privacy-preserving machine learning," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '17. Dallas, Texas, USA: Association for Computing Machinery, Oct. 2017, p. 1175–1191.

[17] Y. Lu, X. Huang, K. Zhang, S. Maharjan, and Y. Zhang, "Communication-efficient federated learning and permissioned blockchain for digital twin edge networks," *IEEE Internet of Things Journal*, vol. 8, no. 4, pp. 2276–2288, Aug. 2021.

[18] Y. Lu, X. Huang, K. Zhang, S. Maharjan, and Y. Zhang, "Communication-efficient federated learning for digital twin edge networks in industrial IoT," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 8, pp. 5709–5718, Aug. 2021.

[19] Y. Jiang, S. Wang, V. Valls, B. J. Ko, W.-H. Lee, K. K. Leung, and L. Tassiulas, "Model pruning enables efficient federated learning on edge devices," Oct 2020.

[20] M. M. Amiri and D. Gunduz, "Federated learning over wireless fading channels," Feb 2020.

[21] J. Mills, J. Hu, and G. Min, "Communication-efficient federated learning for wireless edge intelligence in iot," *IEEE Internet of Things Journal*, vol. 7, no. 7, pp. 5986–5994, 2020.

[22] J. Xu, W. Du, Y. Jin, W. He, and R. Cheng, "Ternary compression for communication-efficient federated learning," *IEEE Transactions on Neural Networks and Learning Systems*, Dec. 2020, early access.

[23] F. Li and B. Liu, "Ternary weight networks," *ArXiv*, vol. abs/1605.04711, Nov. 2016.

[24] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "XNOR-Net: ImageNet classification using binary convolutional neural networks," in *European conference on computer vision*. Springer, Aug. 2016, pp. 525–542.

[25] C. Zhu, S. Han, H. Mao, and W. J. Dally, "Trained ternary quantization," *CoRR*, vol. abs/1612.01064, Dec. 2016. [Online]. Available: http://arxiv.org/abs/1612.01064

[26] J. F. Kurose and K. W. Ross, *Computer Networking: A Top-Down Approach (6th Edition)*, 6th ed. Pearson, 2012.

[27] V.-D. Nguyen, S. K. Sharma, T. X. Vu, S. Chatzinotas, and B. Ottersten, "Efficient federated learning algorithm for resource allocation in wireless IoT networks," *IEEE Internet of Things Journal*, vol. 8, no. 5, pp. 3394–3409, Sep. 2021.

[28] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," Apr. 2014.

[29] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle, "Greedy layer-wise training of deep networks," in *Proceedings of the 19th International Conference on Neural Information Processing Systems*, ser. NIPS'06. MIT Press, Dec. 2006, p. 153–160.

[30] H. Musoff and P. Zarchan, *Fundamentals of Kalman filtering: a practical approach*. American Institute of Aeronautics and Astronautics, Aug. 2009.

[31] O. Ibe, "Markov processes for stochastic modeling: Second edition," *Markov Processes for Stochastic Modeling: Second Edition*, pp. 1–494, May 2013.

[32] G. E. Hinton and R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, pp. 504 – 507, Jul. 2006.

[33] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for Deep Belief Nets," *Neural computation*, vol. 18, no. 7, pp. 1527–1554, Aug. 2006.

[34] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms, Third Edition*, 3rd ed. The MIT Press, Jul. 2009.

[35] S. Laine and T. Aila, "Temporal ensembling for semi-supervised learning," Mar 2017.

[36] C. Lee, K. Cho, and W. Kang, "Directional analysis of stochastic gradient descent via von mises-fisher distributions in deep learning," Nov 2018.

[37] Y. LeCun and C. Cortes, "The MNIST database of handwritten digits," 2010. [Online]. Available: http://yann.lecun.com/exdb/mnist/

[38] G. Cohen, S. Afshar, J. Tapson, and A. Van Schaik, "EMNIST: Extending MNIST to handwritten letters," in *2017 International Joint Conference on Neural Networks (IJCNN)*. Anchorage, AK, USA: IEEE, Mar. 2017, pp. 2921–2926.

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

[40] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, ser. NIPS'12. Red Hook, NY, USA: Curran Associates Inc., Jan. 2012, p. 1097–1105.

[41] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, Dec. 2015. [Online]. Available: http://arxiv.org/abs/1512.03385