FedPoll: A Novel Robustness and Communication-Efficient Aggregation Method

Hamidreza Mehrabian
Chair of Embedded Systems
Ruhr University Bochum
Bochum, Germany
Hamidreza.Mehrabian@ruhr-uni-bochum.de

Akash Kumar
Chair of Embedded Systems
Ruhr University Bochum
Bochum, Germany
Akash.Kumar@ruhr-uni-bochum.de

***Abstract*—Federated learning enables decentralized model training across multiple clients, preserving data privacy by not sharing raw data with a central server while exploiting the variety of data available across clients. Despite its advantages, challenges such as frail privacy, non-independent and non-identical data distributions among clients, and other issues remain unsolved.**

This paper introduces FedWork, a flexible framework for benchmarking federated learning methods. FedWork supports various datasets, aggregation methods, and neural network architectures. We also propose FedPoll, a novel aggregation method that reduces communication overhead and enhances privacy. FedPoll includes two variants: FedPoll-Nearest and FedPoll-MaxMin. Our evaluations show that in non-i.i.d. environments, FedPoll-MaxMin achieves 95% accuracy on the MNIST dataset with partial client participation, outperforming FedAvg, FedAvg-QSGD, FedProx and FedPoll-Nearest, which achieve around 40% with fluctuations. Despite a slight increase in computation time, FedPoll-MaxMin demonstrates significant potential for improving federated learning in challenging settings. Moreover, in a more challenging federated learning scenario, FedPoll-MaxMin achieved acceptable accuracy with ResNet18 on the CIFAR-10 dataset, while FedAvg failed to converge, and SCAFFOLD achieved lower accuracy with greater fluctuations.

***Index Terms*—Federated Learning, FedPoll, FedWork, Aggregation, Robustness, Communication-Efficient**

I. Introduction

In recent years, the proliferation of powerful, compact computational units such as smartphones and various Internet of Things (IoT) devices—including wearable healthcare monitors, smart home systems, and security devices—has created an ideal setting for edge computing and, particularly, edge AI. These devices not only boast significant processing capabilities but also feature a broad array of sensors that generate vast amounts of data. Given the relentless advancements in artificial intelligence, leveraging these developments to improve quality of life has become essential. Consequently, implementing AI on edge devices is no longer just an option but a necessity. Furthermore, training AI models with diverse and extensive datasets is critical for enhancing the models' accuracy and reliability. This robust hardware infrastructure enables the local training of models using readily accessible data.

However, models trained on local datasets often perform optimally only within their specific environments. Accessing a broader range of data poses substantial challenges. A typical solution involves transferring data to centralized servers, which can enhance processing capabilities and accelerate model training. Despite the benefits of increased data diversity, this method raises significant privacy concerns due to the potential for data leakage and misuse during transfer.

To mitigate these challenges, federated learning presents a viable solution. In this model, a central server coordinates the training process but does not access raw data directly. Instead, each participant trains a model on their local dataset and transmits the trained model, not the raw data, back to the server. This approach safeguards privacy and facilitates access to diverse datasets. Consequently, the globally aggregated model, which combines insights from all participant models, is robust and effective across varied local datasets, even those it has not previously encountered.

The concept of federated learning was first introduced by Google in 2016 [16]. Although it resolves many issues, federated learning still faces significant challenges such as communication overhead, hardware diversity, data variability, and privacy and security risks [11], [15]. Communication overhead, a critical concern, stems from the need to synchronize large volumes of model data between the server and clients at the end of each training round. This process can lead to network congestion, particularly over unstable or constrained network connections like WiFi. Hardware diversity further complicates matters, as differences in storage capacity, processor capabilities, and network connections (e.g., WiFi, 5G, 4G) affect the efficiency of participants in the federated learning process. Disparities in hardware and network can lead to inconsistencies in model training and even sudden disconnections [9]. Additionally, statistical heterogeneity, which arises from variations within datasets, remains an inherent challenge that must be managed.

Privacy and security remain paramount in federated learning. Although this approach obviates the need to transmit raw data to a central server, it is still possible to infer sensitive user information from the models sent to the server [22]. Furthermore, the system is vulnerable to various types of attacks. Data Poisoning, for instance, involves manipulating the trained local model, while Model Poisoning targets the integrity of client data, both of which are potent threats posed by malicious entities [15].

To counteract the challenge of statistical heterogene-

This work is supported by the Deutsche Forschungsgemeinschaft (DFG) under the X-ReAp project (Project number 380524764).

ity—stemming from the diverse and non-i.i.d. nature of distributed private local datasets—various strategies have been developed. One notable method is FedProx, introduced by Li et al. [10]. This technique incorporates an additional penalty term into the loss function during the local training phase at each node. This penalty reflects the divergence between the local model and the globally aggregated model. Thus, during training, clients strive to minimize not only their individual model's error but also the disparity between their model and the global standard. This dual optimization helps align the local updates more closely with the global model, improving overall system coherence and effectiveness. In addition, Karimireddy et al. [7] proposed a method called SCAFFOLD to address variance among clients in federated learning. This approach mitigates client drift during local updates by using control variates, which correct the misalignment between local and global models. SCAFFOLD leverages similarities in client data, resulting in more consistent updates and faster convergence. Consequently, fewer communication rounds are needed to achieve convergence, enhancing the efficiency of the federated learning process. However, it requires updating global control variates by sending the client control variates model along with the client model at each round. This necessity effectively and approximately doubles the amount of data exchanged between clients and the server per round, as the control variates model has the same shape as the client model.

To reduce communication overhead, numerous strategies have been developed for efficiently transferring models or gradients. Konečný et al. [8] detailed primary methods for decreasing model size during synchronization. They explored two general strategies: Structured updates, which involve sending structurally simplified models from clients to the server, and Sketched updates, which entail sending compressed models. Striking a balance between accuracy and reduced communication size is a constant trade-off. Reisizadeh et al. [18] introduced a technique where only the quantized gradients of trained models from a subset of clients are sent periodically, rather than at the end of each training round. WANG et al. [20] proposed a method named CMFL that aims to reduce communication demands by transmitting only those gradients where the frequency of sign changes between the current and previous gradients exceeds a specific threshold. This selective transmission significantly lowers the communication load.

A prominent approach in enhancing communication efficiency in federated learning is through Sketched updates, especially via quantization methods. Various types of quantization, such as Uniform Quantization and Probabilistic Quantization [8], are employed to streamline data transmission [21]. A recent notable quantization technique, QSGD, was introduced by Alistarh et al. [1]. In QSGD, gradients are quantized using randomized rounding to discrete values. Unlike deterministic methods, which round gradients to the nearest discrete level, QSGD uses randomized rounding, assigning each gradient component to one of several predefined levels. This randomization mitigates quantization error and potentially enhances the convergence properties of SGD, particularly where traditional quantization might introduce bias.

The importance of privacy and security in federated learning has consistently been a focal point. Liu et al. [13] conducted a comprehensive survey on privacy-preserving strategies during the aggregation stage of federated learning. Their findings highlight several key techniques for maintaining privacy: One-time Pad, which utilizes a random key at the beginning of communications; Homomorphic Encryption, enabling arithmetic operations like averaging on encrypted models without decryption; Secure Multi-Party Computation, which allows multiple parties to collaboratively compute on their private data while preserving confidentiality and without disclosing individual inputs; Differential Privacy, which involves adding noise to the model so that the outcomes of operations such as averaging remain unchanged; Blockchain, incorporating smart contracts and cryptocurrency benefits into federated learning; and Trusted Execution Environments, secure areas within main processors like ARM-TrustZone that protect sensitive data and code in an isolated and trusted space.

Additionally, Lyu et al. [14] have provided an extensive survey on recent threats to federated learning, including potential attacks and defensive measures. Geyer et al. [5] introduced methodologies for incorporating Gaussian noise at the client level, a technique that bolsters privacy while preserving the accuracy of the model.

In this paper, we introduce a lightweight federated learning framework designed for research purposes, named FedWork. Additionally, we propose a novel aggregation method called FedPoll, specifically developed to enhance communication efficiency while improving security and privacy within the federated learning process.

Section II is dedicated to discussing proposed federated learning frameworks. In this section, we introduce several existing frameworks, unveil FedWork, outline its key features, and depict its architecture. Section III provides a comprehensive overview of FedPoll, our innovative aggregation method. This section delves into the federated learning procedure, with a specific focus on the aggregation step, and provides a detailed explanation of the FedPoll method. Additionally, we introduce two specific approaches within this concept and detail the FedPoll algorithm. Finally, in Section IV, we present comparative analyses of FedPoll, illustrated through various figures.

II. FedWork

The expanding landscape of federated learning has highlighted the need for frameworks that can effectively assess and analyze different methodologies. Several open-source federated learning frameworks, designed to cater to research needs, are currently available. Notable examples include TensorFlow Federated (TFF) by Google [3], PySyft [19], Federated AI Technology Enabler (FATE) [12], LEAF (A Benchmark for Federated Settings) [4], OpenFL (Open Federated Learning) [17], Flower (A Friendly Federated Learning Framework) [2], and FedML [6].

Each of these frameworks serves a unique purpose: TFF enhances the scalability of TensorFlow applications; PySyft focuses on privacy and security; FATE is tailored for federated AI ecosystems with robust security features; LEAF functions as a benchmarking tool for federated algorithms; OpenFL

facilitates secure and collaborative learning across different entities; FedML offers a comprehensive platform that supports a wide range of algorithms for researchers and developers; and Flower provides a user-friendly, scalable, and framework-agnostic platform that is compatible with various machine learning libraries. Among these, Flower, FedML, and LEAF originate from academic initiatives, while the rest are industry developments.

In this paper, we introduce FedWork, our newly developed framework designed to assess and evaluate the FedPoll method. During our exploration of federated learning, we recognized the necessity for an ultra-lightweight, highly configurable framework. FedWork, which is under 1MB in size, meets this need by offering exceptional flexibility and extensibility. It facilitates rapid implementation and comparative analysis of various strategies or ideas, and it supports the creation of customizable reports. Developed as an open-source tool, FedWork is specifically designed for benchmarking and comparison, employing PyTorch as its foundational machine learning library.

In FedWork, the entirety of the assessment functionalities is configured through an XML file. This configuration encompasses dataset settings (including type and specifics), server and virtual client settings (detailing participating clients), comparison methods along with their specific configurations, and the configurations for output figures.

FedWork includes a TCP network layer that is responsible for managing packet framing and queuing, efficiently handling the transmission and reception of large packets in segmented chunks.

Additionally, FedWork features a unique client negotiation mechanism. Each client is added to a client pool, and at the start of each federated round, the federated learning method determines the participation of clients based on predefined criteria.

Prior to initiating the learning process, FedWork generates datasets as specified in the XML file, designed in a non-i.i.d manner. "Non-i.i.d" refers to datasets that are not "independent and identically distributed," a common scenario in federated learning where each participant maintains a distinct, private dataset. This configuration introduces complexities such as data overlap, class imbalance, statistical heterogeneity, label distribution skew, and feature distribution skew.

The dataset generator in FedWork can create datasets with specified non-i.i.d. levels, expressed as percentages, while also allowing for heterogeneous dataset sizes. This parameter controls how much the label distributions deviate from uniformity. A higher non-i.i.d. level leads to more variability in the class distribution among clients. Figure 1 demonstrates this using the MNIST dataset across a range of non-i.i.d. levels.

FedWork is equipped with several pre-implemented federated learning methods, including FedAvg, FedProx, FedPAQ, Quantized FedAvg, and FedPoll, which we will explore in detail in the following section. Adding a new federated learning method to the framework is simplified through a modular design. Developers can easily integrate new methods by defining a class that inherits from a parent class and implementing its abstract functions, such as aggregate, start training, and select clients to train.

The framework also supports a variety of pre-implemented neural network architectures, such as FeedForwardNet1 (two layers), FeedForwardNet2 (three layers), and ResNet18. Each architecture is encapsulated within a class, with configuration variables dynamically assigned based on the settings in the main XML configuration file prior to the assessment process. Introducing a new neural network architecture involves creating a class file for the neural network, allowing for seamless expansion.

For benchmarking and monitoring performance, FedWork employs a profiler service that measures the duration between two code points or tracks changes in specific variables. The framework inherently records many essential parameters, and extending its monitoring capabilities to include additional parameters is straightforward through method extensions.

Additionally, FedWork incorporates a robust logger service that offers multiple levels of logging and supports various output types, such as files and network logs, configurable through the XML file.

A. Architecture

FedWork is designed for flexibility while maintaining a straightforward structure.

The architecture of FedWork revolves around a core module that interprets the XML configuration file and orchestrates subsequent processes. Initially, this core engages a non-i.i.d dataset generator to construct the required datasets according to the predefined specifications. These specifications include the type of dataset (e.g., CIFAR10, MNIST, FashionMNIST), the level of non-i.i.d dispersion, size heterogeneity, and other relevant parameters. The datasets thus generated are stored as discrete files.

Following dataset generation, the architecture simulates virtual clients (representing federated learning nodes) on the server machine. The core then proceeds to assemble the necessary federated learning methods and constructs the neural network architecture based on the specifications detailed in the XML file, such as the number of nodes in hidden layers.

The federated learning process is initiated and continues through the specified rounds of the method, with performance and operation data being systematically captured and stored by the profiler service. This process is replicated for all methods delineated in the configuration file, with each method's profiler producing a detailed report.

Ultimately, the core compiles benchmarks and generates comprehensive reports as outlined in the configuration file, which are presented in figure formats. The structural components of FedWork are depicted in Figure 2. You can access this framework here: https://github.com/hamidrm/FedWork.

III. FedPoll

In this section, we introduce a novel aggregation method called FedPoll, which utilizes a polling mechanism among clients to select appropriate values for each corresponding element in the global model. This method not only reduces the size of communication data, thereby decreasing communication overhead, but it also enhances privacy and security.

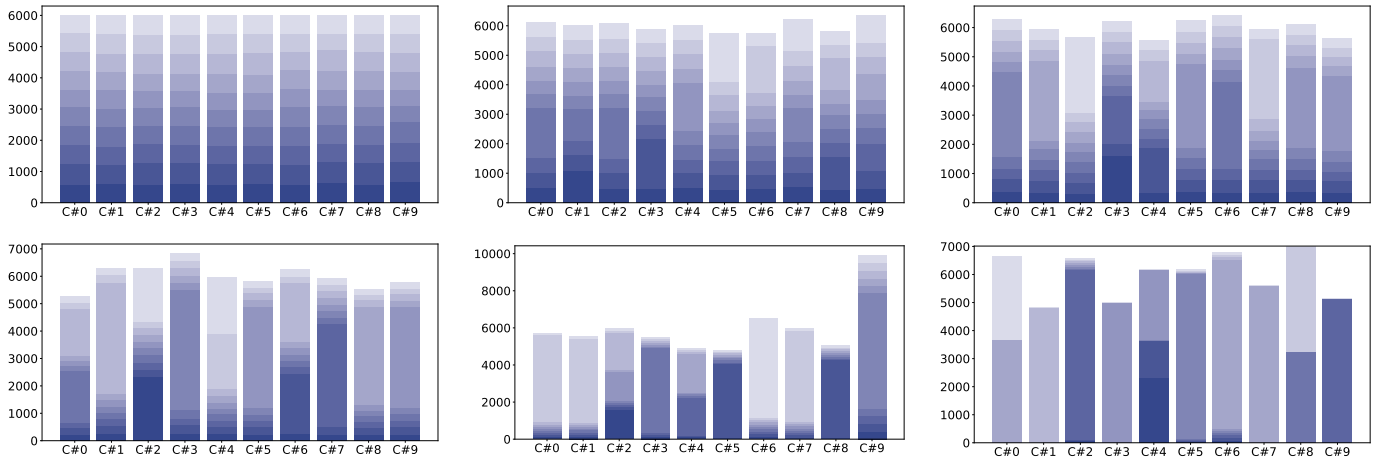


Fig. 1: Visualizing the impact of non-i.i.d. levels on data distribution and sample size across clients in FedWork (0%, 20%, 40% on the first row; 60%, 80%, and 100% on the second row, from left to right)

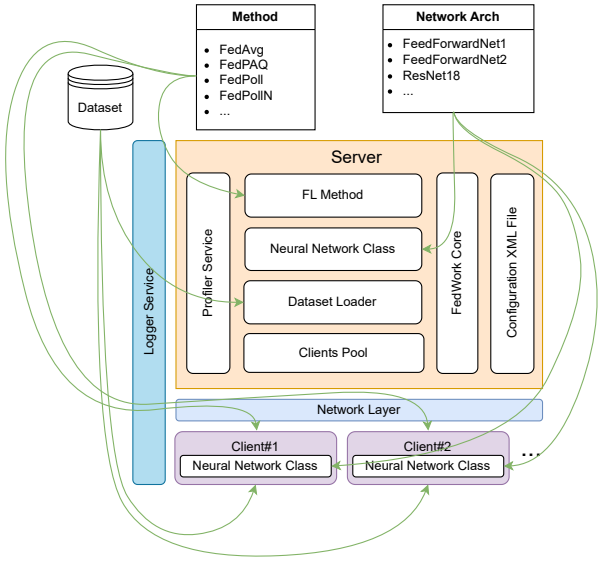


Fig. 2: Blocks of the FedWork Framework

We will discuss the specific mechanisms of FedPoll and propose two approaches based on the polling process. The implementation of this method within the FedWork framework and the evaluation results will be presented in the following section.

A. Main Idea

FedPoll aggregates client models based on their choices rather than directly using the raw model data. A significant challenge in federated learning is the communication overhead and the volume of data that must be transmitted from clients to the server. FedPoll addresses this by transmitting pointers to choices instead of the actual model values, significantly reducing the data size and making it configurable.

Moreover, the non-i.i.d nature of client datasets in federated learning often hampers the convergence of the global model. The FedPoll method demonstrates notable improvements in mitigating this issue. Additionally, as previously discussed, client privacy is at risk in conventional federated learning setups. By using pointers rather than actual data values, FedPoll enhances both the security and privacy of client data.

In federated learning, the overarching goal is to minimize a sum of non-convex neural network objectives across various clients. Assuming each client has an equal sample size, the optimization problem can be typically formulated as:

$$\min_{\theta} f(\theta), \quad f(\theta) := \frac{1}{N} \sum_{i=1}^{N} F_i(\theta) \tag{1}$$

This formulation seeks the optimal parameter set, θ, which best fits the global model. Each local objective, F_i, for node i is defined as the expected value of the loss function over its local sample distribution:

$$F_i(\theta) := \mathbb{E}_{\xi \sim \mathcal{D}_i}[\mathcal{L}(\theta, \xi)] \tag{2}$$

In the equation above, ξ represents a local dataset sample from a distribution $\mathcal{D}_i$, which typically adheres to an unknown distribution due to the non-i.i.d nature of the data. $\mathcal{L}$ denotes the loss function utilized by the client.

In FedAvg and similar federated learning methodologies, the process typically begins with the server initializing the global model. This iterative process involves several steps until specific criteria are satisfied, summarized as follows:

1) **Global Model Distribution**: The server sends the global model to all participating clients.
2) **Model Initialization**: Clients initialize their models using the received global model.
3) **Local Training**: Clients train on their local datasets for a predetermined number of epochs using a specific optimizer.
4) **Model Submission**: Clients send their trained models (or the gradients) back to the server.

5) **Model Aggregation**: The server aggregates the received models, typically using averaging or another method.
6) **Convergence Check**: The server evaluates whether the predefined criteria have been met. If not, the iteration cycle repeats.

Each iteration of these steps is considered a round, which repeats upon the completion of the previous iteration or after a specified time interval.

Within this ecosystem, each client trains its model using an optimizer such as Stochastic Gradient Descent (SGD) over a designated number of epochs. The optimization, particularly when applied to mini-batches of data, is known as mini-SGD. The optimization formula for training the model using SGD is expressed as:

$$\theta_{t+1}^{j} = \theta_{t}^{j} - \eta \nabla f(\theta_{t}^{j}) \tag{3}$$

where θ_{t+1}^{j} and θ_{t}^{j} are the model parameters at times $t+1$ and t respectively, η is the learning rate, and $\nabla f(\theta_{t}^{j})$ is the gradient of the loss function with respect to the model parameters at time t.

After training for a designated number of epochs, the clients send their trained models to the server. The aggregation step, crucial for updating the global model, involves combining all the clients' models. In FedAvg, which is the foundational method in federated learning, the global model is updated by averaging the trained models received from the clients in each round. The mathematical formulation for updating the global model in FedAvg is given by:

$$\theta_{t+1}^{G} = \frac{1}{m} \sum_{j=1}^{N} n_j \theta_j \tag{4}$$

Here, θ_{t+1}^{G} represents the global model parameters at time $t+1$, while θ_j denotes the trained model received from the j-th client after a certain number of epochs. The symbol m represents the total number of data points across all clients, N is the number of clients, and n_j is the number of data points held by the j-th client.

In FedPoll, the process initiates with the server setting up a global model and distributing it to the clients. Each client then optimizes this model based on their local dataset and sends the trained model back to the server. In the initial round, the server uses an averaging technique similar to FedAvg to compute the first version of the global model. In subsequent rounds, the server introduces a new method where it provides multiple suitable random values C for each model element. The clients select from these values based on their optimized models. Both clients and server synchronize using the same random set for each round and for each network layer, with synchronization in the first round facilitated through the sharing of an initial seed.

Additionally, the server sends the global model along with the appropriate radius for each layer in each round (figure 5). This radius determines the maximum allowable deviation between the newly trained model values and the previous ones across all clients for each network layer. The significance and impact of this radius will be discussed in detail in the following section. In this method, clients do not send the gradients of their trained models. Instead, they send the selected indices of values from the random set. This selection is conducted using the function $g(C, i, \theta_{t+1}^{j})$, where j is the index of the client. If this function returns true, the i-th index of the random set C will be chosen. The set of selected indices, V, is then sent to the server. The server aggregates the clients' models using a function h to compute the global model θ_{t+1}^{G}. In this paper, we propose two approaches for the g and h functions, which will be discussed in the following sections.

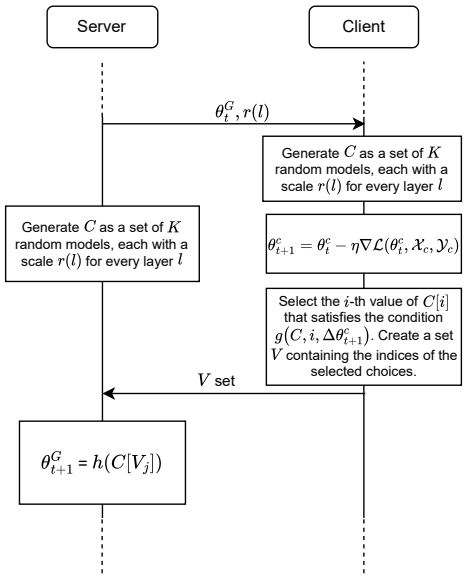


Fig. 3: FedPoll Round Process

Additionally, FedPoll presents a common random set that divides the probable boundary of the gradient changes of clients in each round into K segments. These values are random, and clients select one based on their trained model. This aspect of FedPoll employs a specific type of quantization.

Clients attempt to represent their trained model using one of the random values, selecting the most appropriate one (Figure 4). The random set of candidate values, $C(i)$, adheres to a uniform distribution. In the next section, we will discuss this method in more detail.

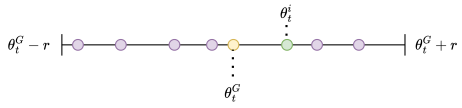


Fig. 4: Random points with current optimum value for client i and global optimum value

B. Principles

According to the FedPoll method, the difference between the newly trained model and the prior trained model for each parameter in each client model at the end of the round is

bounded by the value r. Additionally, the new gradient can be expressed by adding the value $C_t[V_j]$ to a specific error value e_t^j for the j-th client, compensating for the difference between the actual gradient value and $C_t[V_j]$. Thus, we can establish some primary assumptions as follows:

$$\begin{aligned} &|\theta_{t+1}^j - \theta_t^j| \leq r_{t+1}^j \\ &\eta \nabla f(\theta_t^j) = C_t[V_j] + e_t^j \end{aligned} \tag{5}$$

Regarding (3), we can rewrite (5) as:

$$\theta_t^j - \theta_{t+1}^j = C_t[V_j] + e_t^j \tag{6}$$

$$|C_t[V_j] + e_t^j| \leq r_{t+1}^j \tag{7}$$

We assume $C \sim \mathcal{U}(-r, r)$ as the candidate set, which is populated with random values. In a non-i.i.d. federated learning ecosystem (which closely mirrors real-world conditions), θ_{t+1}^j, the model obtained after e epochs of training by the j-th client on its local dataset, follows an undetectable and unknown distribution. This is due to the completely unpredictable nature of the local datasets, which cannot be associated with any standard type of distribution.

If the g function selects the nearest value to the actual gradient θ_{t+1}^j, $C[V_j]$ will also adhere to an unknown distribution. This probabilistic environment presents a significant challenge in modeling the changes in clients' gradients.

The probability of choosing a suitable value can be determined by the following equation:

$$P(|\eta \nabla f(\theta_t^j) - C_t[V_j]| < \epsilon_m) = 1 - \left(1 - \frac{\epsilon_m}{r}\right)^K \tag{8}$$

We know that if K, the number of random candidates, is sufficiently large, we can assume e_t^j (where $\epsilon_m = \max_j e_t^j$) can be made to approach zero. This is because, with an increasing number of options for clients to choose from, according to the role of the g function, the probability of $C_t[V_j]$ approaching the actual gradient $\eta \nabla f(\theta_t^j)$ increases as well.

We need to find an appropriate approximation for the optimum value of $r(l)$.

If we sum (6) across all clients, we get:

$$\sum_{j=1}^{N} (\theta_t^j - \theta_{t+1}^j) = \sum_{j=1}^{N} (C_t[V_j] + e_t^j) \tag{9}$$

According to (4) and (3), we can rewrite (10) as:

$$\theta_t^G - \theta_{t+1}^G = \frac{1}{N} \sum_{j=1}^{N} (C_t[V_j] + e_t^j) \tag{10}$$

$$|\Delta \theta_{t+1}^G| = \frac{1}{N} \left| \sum_{j=1}^{N} (C_t[V_j] + e_t^j) \right| \tag{11}$$

We also know that:

$$\frac{1}{N} \left| \sum_{j=1}^{N} (C_t[V_j] + e_t^j) \right| \leq \frac{1}{N} \sum_{j=1}^{N} \left| C_t[V_j] + e_t^j \right| \tag{12}$$

Thus, according to (7):

$$\frac{1}{N} \left| \sum_{j=1}^{N} (C_t[V_j] + e_t^j) \right| \leq r_{t+1}^j \tag{13}$$

Finally, we can bound r for each parameter in the models via:

$$|\Delta \theta_{t+1}^G| \leq r_{t+1}^j \tag{14}$$

We can find r as $\max_j r_{t+1}^j$ across all clients. This gives us:

$$|\Delta \theta_{t+1}^G| \leq r$$

Additionally, we can cluster a batch of parameters of the layers in the neural networks to mitigate the number of r parameters. Thus, the minimum possible value for r will be:

$$r(l) = \overline{|\Delta \theta_t^G(l)|} \tag{15}$$

We add a constant value ϵ to allow some drift for the next round gradient changes:

$$r(l) = \overline{|\Delta \theta_t^G(l)|} + \epsilon \tag{16}$$

We will discuss the results of this approach in the next section. We will present two different approaches for the FedPoll functions in the following subsections.

FedPoll Nearest Based

This approach provides a straightforward method for determining the functions g and h. The function g selects the candidate that is closest to the actual value. The function h, on the other hand, computes a simple average of the candidates based on data from N clients. It takes into account the total number of data points across all clients (m) and the number of data points specific to client j (n_j).

$$g(C, i, \theta_t) : i == \arg\min_m \{|C[m] - \theta_t|\}$$

$$h(w) : \frac{1}{m} \sum_{j=1}^{N} n_j w_j$$

FedPoll Max-Min

In this approach, we select the first candidate that exceeds the actual value, introducing a positive bias to the estimation. First, we sort the candidates in ascending order on both the client and server sides. The client then chooses the candidate based on the first value in the sorted set C that is greater than θ_t.

We define the function g such that it returns a Boolean value indicating whether a given index i is the expected index where the condition $\theta_t < C_m$ first holds in the sorted list of candidates C. Specifically, the function is defined as:

$$g(C, i, \theta_t) : i == \arg\min_m \{C[m] > \theta_t\}$$

In this approach, the h function does not use simple averaging. Instead, it assesses each of the received indices from all clients. Using the sorted set C and re-indexing it according to the new indices, we find the minimum index across all clients that is greater than zero, denoted as a. We also find the maximum

index across all clients that is less than N, denoted as b. Thus, we can write h as follows:

$$h(w) : \frac{C_a + C_b}{2}$$

In special cases where all clients select the index zero or N, we use the minimum and maximum values within the candidates, respectively.

With these definitions, we use the midrange, which is the middle value between the maximum and the minimum approximations represented by the clients. By aggregating clients' models using midrange instead of average, we choose the middle value of optimal points for each model's parameter across all clients at each communication round. If clients use IID data, we can estimate a Gaussian distribution across clients for each model's parameter. In this way, the expected value of the average of the parameters approaches the mean value, and after an appropriate number of rounds, we will have an acceptable global optimum point.

We propose using the midrange, which will achieve a global optimum due to the model distribution. Although the midrange will increase the variance over rounds, it will take more rounds to reach a certain optimum value. However, we can reduce communication overhead significantly. Additionally, by using midrange instead of averaging, we pay more attention to distinct clients (isolated islands) who have completely different datasets in non-i.i.d environments (Figure 6).

Adopting midrange aggregation indeed shifts the objective of federated learning from minimizing a loss function through averaging to finding a central point between extremes.

For the r value, we use (16).

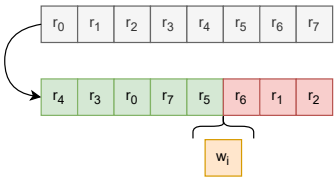


Fig. 5: Choosing an appropriate index of sorted candidates, denoted as W_i, represents the optimal model parameter. The green random values are less than W_i. Consequently, the client will use 6 as this parameter to send.

C. Algorithm

Regarding the FedPoll process, we can represent its algorithm in Algorithm 1. This algorithm uses the g and h functions discussed in the previous subsections.

IV. Evaluations

All our evaluations were conducted using the FedWork framework. We assessed the performance of our proposed method, FedPoll, in comparison with FedProx, SCAFFOLD, standard FedAvg, and quantized FedAvg. Each evaluation involved 10 clients with a medium level of non-i.i.d. data distribution (50% in FedWork). The neural network architecture used was a simple fully connected feed-forward linear network with one hidden layer (784x1024x10). In each round, 5 randomly selected clients (out of the 10 available) contributed to training the model over 3 epochs across 200 rounds. The results for loss and accuracy are shown in Figures 7a and 7b. FedAvg, FedAvg-QSGD, FedProx, and FedPoll-Nearest achieved approximately 40% accuracy with significant fluctuations, while FedPoll-MaxMin reached 95% and SCAFFOLD achieved 96%. In these evaluations, ϵ for both FedPoll methods was set at 0.01 and K at 8, indicating a need for 3-bits per parameter. FedAvg-QSGD was configured for 7 levels of quantization, and μ for FedProx was set to 1. The compression ratios for FedPoll-Nearest, FedPoll-MaxMin, and FedAvg-QSGD were 10.6, 10.6, and 8, respectively. As shown in Figure 7b, the computational overhead for FedPoll-MaxMin was slightly higher than that of FedProx. However, over 200 rounds, FedPoll-Nearest and SCAFFOLD required more time than FedPoll-MaxMin. It is important to note that, while FedPoll's accuracy is 1% lower than SCAFFOLD's, FedPoll uses only 3 bits (index of the selected candidate) per parameter compared to SCAFFOLD's 64 bits (32 bits for the model + 32 bits for control variates). In our next evaluation (Figure 8), we tested these methods under more challenging conditions using the CIFAR-10 dataset, which was distributed among all clients with a moderate level of non-i.i.d. For this evaluation, we employed the ResNet18 architecture. As before, 5 out of 10 clients contributed to training the model over 3 epochs per round, with K set to 8, indicating a need for 3 bits per parameter. The evaluation included FedPoll-MaxMin, SCAFFOLD, and FedAvg.

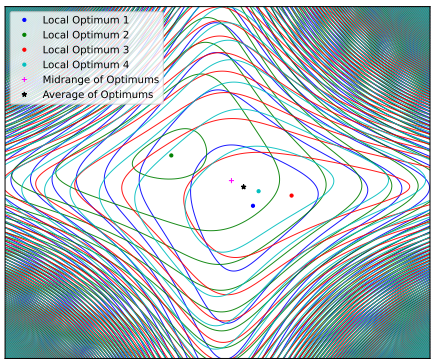


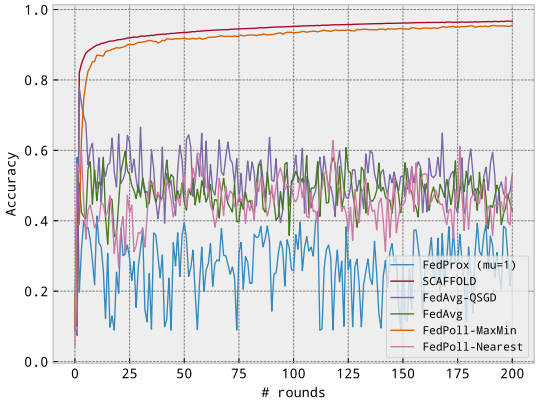
Fig. 6: Imaginary optimal values for four clients' convex functions and the server-calculated optimal point using the Midrange and Average methods.

The accuracy of the FedAvg method stagnated at its initial level, never exceeding 10% on the CIFAR-10 dataset, which consists of 10 classes. In contrast, SCAFFOLD initially demonstrated a rapid increase in accuracy compared to FedPoll-MaxMin but exhibited significant fluctuations and showed less improvement after first 40 rounds, ultimately reaching 72%. Meanwhile, FedPoll-MaxMin steadily improved, achieving approximately 85% accuracy after 100 rounds with $\epsilon = 10^{-3}$.

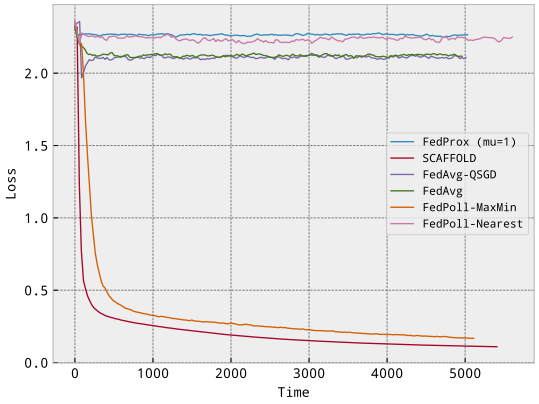
Algorithm 1 FedPoll Algorithm

Server Variables: $r[], FirstAggregation$
procedure FEDPOLLSERVER($\theta_G, \Delta\theta_C[]$)
  if *FirstAggregation* = True **then**
    FirstAggregation ← False
    $\theta_G \leftarrow$ FEDAVG($ClientModels$)
    return θ_G
  end if
  for l **in** $\theta_G.Layers$ **do**
    $C \leftarrow$ empty list
    for $k = 1$ **to** K **do**
      $R \leftarrow$ RANDOMLIKE($\Delta\theta_G[l], -r[l], r[l]$) ▷ Generate R sets like $\Delta\theta_G.Layers[l]$ within radius $r[l]$
      Append R **to** C
    end for
    $T \leftarrow$ **apply** $\Delta\theta_C$ **based on** C
    $NewLayer \leftarrow$ HFUNCTION($T + \theta_G.Layers[l]$)
    $r[l] \leftarrow |NewLayer - l| + \epsilon$
    $\theta_G[l] \leftarrow NewLayer$
  end for
end procedure

Client Variables: $r[], e, FirstAggregation$
procedure FEDPOLLCLIENT(θ_G)
  $\theta \leftarrow \theta_G$
  for *epoch* = 1 **to** e **do**
    $\theta \leftarrow \theta - \eta\Delta f(\theta, \mathcal{X}, \mathcal{Y})$
  end for
  $\Delta\theta \leftarrow$ updates of the trained model
  if *FirstAggregation* = True **then**
    FirstAggregation ← False
    return $\Delta\theta$
  end if
  $C \leftarrow$ empty list
  for $k = 1$ **to** K **do**
    $R \leftarrow$ RANDOMLIKE($\Delta\theta_G[l], -r[l], r[l]$) ▷ Generate R sets like $\Delta\theta_G.Layers[l]$ within radius $r[l]$
    Append R to C
  end for
  for *index*, *parameter* **in** $\Delta\theta$ **do**
    $\Delta\theta[index] \leftarrow k$ **where** GFUNCTION($C[index], k, \Delta\theta$) **is True**
  end for
  return $\Delta\theta$
end procedure



(a) Accuracy



(b) Loss value

Fig. 7: Methods Comparison

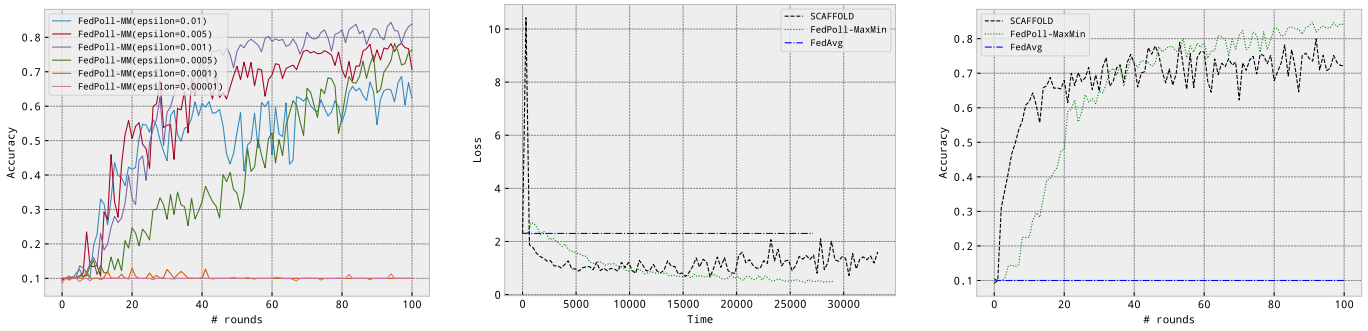
Additionally, our experiments with varying ϵ values highlighted their substantial impact on the accuracy of FedPoll-MaxMin. A very low ϵ value resulted in minimal and stagnant accuracy. A larger ϵ value led to a slow but steady increase in accuracy. Conversely, an even larger ϵ value caused greater fluctuations in accuracy without yielding significant improvements compared to other ϵ values.

V. CONCLUSION

We proposed a flexible and modular federated learning framework called FedWork, which supports various datasets, aggregation methods, and neural network architectures. This lightweight and straightforward framework is designed for benchmarking different approaches and ideas in the federated learning domain. The flexibility of FedWork is largely attributed to its XML configuration file, which allows for comprehensive settings adjustments in federated learning experiments.

Additionally, we introduced a new aggregation method called FedPoll, which functions similarly to a quantization method in a non-uniform data distribution environment. FedPoll reduces overhead and bottlenecks by decreasing the size of data sent from low-resource clients to the server. Moreover, by incorporating differential privacy features and avoiding the transmission of values directly related to the trained model, it enhances privacy and security. We proposed two approaches within this strategy: FedPoll-Nearest and FedPoll-MaxMin.

Our experiments demonstrate that, in a non-i.i.d. environment with only a subset of clients participating, the standard FedAvg algorithm struggles to converge. In an initial evaluation with the non-i.i.d. MNIST dataset, employing a convex architecture (a linear feedforward network with one hidden layer) and partial client contributions, FedPoll-MaxMin attained 95% accuracy, while SCAFFOLD reached



(a) Different r values (Clients: 3/10) (b) Comparison of loss (Clients: 5/10) (c) Comparison of accuracy (Clients: 5/10)

Fig. 8: Evaluation on CIFAR10 and ResNet18

96%. FedAvg, FedAvg-QSGD, and FedPoll-Nearest, however, only achieved around 40% accuracy with notable fluctuations. Although FedPoll-MaxMin incurs slightly higher computation time compared to FedAvg (but less than SCAFFOLD), its performance highlights its potential for improving federated learning outcomes in challenging environments.

In our second evaluation using ResNet18 with the CIFAR-10 dataset, a non-convex optimization problem, SCAFFOLD achieves approximately 72% accuracy, though with significant fluctuations. In contrast, FedPoll-MaxMin delivers a more stable accuracy of 85%.

It's worth noting that FedPoll requires only an index (3 bits in our evaluations) for each parameter, rather than transmitting raw data. Conversely, SCAFFOLD needs a floating-point number for client control variates in addition to a floating point for each parameter. Moreover, the noisy nature of FedPoll's data and the lack of access to the shared seed during the initial communication round make it difficult for a man-in-the-middle attacker to infer the transmitted model. Additionally, a malicious client would struggle to alter the global model due to the bounded values of candidate parameters.

REFERENCES

[1] D. Alistarh, D. Grubic, J. Li, R. Tomioka, and M. Vojnovic. Qsgd: Communication-efficient sgd via gradient quantization and encoding. *Advances in neural information processing systems*, 30, 2017.

[2] D. J. Beutel, T. Topal, A. Mathur, X. Qiu, J. Fernandez-Marques, Y. Gao, L. Sani, K. H. Li, T. Parcollet, P. P. B. de Gusmão, et al. Flower: A friendly federated learning research framework. *arXiv preprint arXiv:2007.14390*, 2020.

[3] K. Bonawitz, H. Eichner, W. Grieskamp, D. Huba, A. Ingerman, V. Ivanov, C. Kiddon, J. Konečnỳ, S. Mazzocchi, B. McMahan, et al. Towards federated learning at scale: System design. *Proceedings of machine learning and systems*, 1:374–388, 2019.

[4] S. Caldas, S. M. K. Duddu, P. Wu, T. Li, J. Konečnỳ, H. B. McMahan, V. Smith, and A. Talwalkar. Leaf: A benchmark for federated settings. *arXiv preprint arXiv:1812.01097*, 2018.

[5] R. C. Geyer, T. Klein, and M. Nabi. Differentially private federated learning: A client level perspective. *arXiv preprint arXiv:1712.07557*, 2017.

[6] C. He, S. Li, J. So, X. Zeng, M. Zhang, H. Wang, X. Wang, P. Vepakomma, A. Singh, H. Qiu, et al. Fedml: A research library and benchmark for federated machine learning. *arXiv preprint arXiv:2007.13518*, 2020.

[7] S. P. Karimireddy, S. Kale, M. Mohri, S. Reddi, S. Stich, and A. T. Suresh. Scaffold: Stochastic controlled averaging for federated learning. In *International conference on machine learning*, pages 5132–5143. PMLR, 2020.

[8] J. Konečnỳ, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon. Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492*, 2016.

[9] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith. Federated learning: Challenges, methods, and future directions. *IEEE Signal Processing Magazine*, 37(3):50–60, 2020.

[10] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith. Federated optimization in heterogeneous networks. *Proceedings of Machine learning and systems*, 2:429–450, 2020.

[11] B. Liu, N. Lv, Y. Guo, and Y. Li. Recent advances on federated learning: A systematic survey. *Neurocomputing*, page 128019, 2024.

[12] Y. Liu, T. Fan, T. Chen, Q. Xu, and Q. Yang. Fate: An industrial grade platform for collaborative learning with data protection. *Journal of Machine Learning Research*, 22(226):1–6, 2021.

[13] Z. Liu, J. Guo, W. Yang, J. Fan, K.-Y. Lam, and J. Zhao. Privacy-preserving aggregation in federated learning: A survey. *IEEE Transactions on Big Data*, pages 1–20, 2022.

[14] L. Lyu, H. Yu, X. Ma, C. Chen, L. Sun, J. Zhao, Q. Yang, and P. S. Yu. Privacy and robustness in federated learning: Attacks and defenses. *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–21, 2022.

[15] P. M. Mammen. Federated learning: Opportunities and challenges. *arXiv preprint arXiv:2101.05428*, 2021.

[16] 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*, pages 1273–1282. PMLR, 2017.

[17] G. A. Reina, A. Gruzdev, P. Foley, O. Perepelkina, M. Sharma, I. Davidyuk, I. Trushkin, M. Radionov, A. Mokrov, D. Agapov, et al. Openfl: An open-source framework for federated learning. *arXiv preprint arXiv:2105.06413*, 2021.

[18] A. Reisizadeh, A. Mokhtari, H. Hassani, A. Jadbabaie, and R. Pedarsani. Fedpaq: A communication-efficient federated learning method with periodic averaging and quantization. In *International conference on artificial intelligence and statistics*, pages 2021–2031. PMLR, 2020.

[19] T. Ryffel, A. Trask, M. Dahl, B. Wagner, J. Mancuso, D. Rueckert, and J. Passerat-Palmbach. A generic framework for privacy preserving deep learning. *arXiv preprint arXiv:1811.04017*, 2018.

[20] L. WANG, W. WANG, and B. LI. Cmfl: Mitigating communication overhead for federated learning. In *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, pages 954–964, 2019.

[21] Z. Zhao, Y. Mao, Y. Liu, L. Song, Y. Ouyang, X. Chen, and W. Ding. Towards efficient communications in federated learning: A contemporary survey. *Journal of the Franklin Institute*, 360(12):8669–8703, 2023.

[22] L. Zhu, Z. Liu, and S. Han. Deep leakage from gradients. *Advances in neural information processing systems*, 32, 2019.