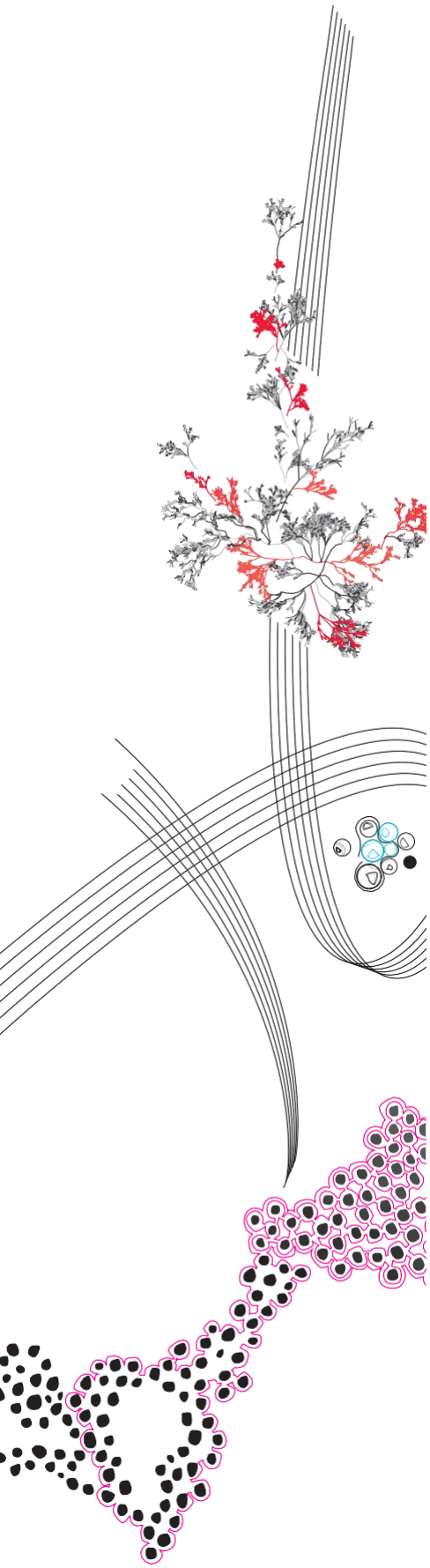

MSc Computer Science

FedNIP: A Statistical Heterogeneity Aware Dynamic Ranking Algorithm for Federated Learning

Sjoerd Zagema

Supervisor: Dr.ir. Alex Chiumento
Committee member: Prof. dr. Paul Havinga
Committee member: Dr.ir. Andrea Continella

April, 2024

Department of Electrical Engineering,
Mathematics and Computer Science
Pervasive Systems Research Group

UNIVERSITY OF TWENTE.

FedNIP: A Statistical Heterogeneity Aware Dynamic Ranking Algorithm for Federated Learning

ABSTRACT

Federated Learning (FL) is a cutting-edge approach to Machine Learning (ML) that allows for the decentralized training of models, without the need for centralizing the raw data. This ensures the privacy of the client, as the actual data never leaves the device. However, a major challenge in FL is that clients often have significant differences in their local data distributions, which leads to a suboptimal convergence speed and decreased accuracy. To address this issue, a novel FL algorithm called Federated Non-IID Performance (FedNIP) is proposed.

FedNIP is a dynamic ranking-based exploration and algorithm, prioritizing clients based on their impact on the global model. Unlike other FL algorithms, FedNIP dynamically updates client performance and ranks clients to prevent bias in the training. Clustering is used to group clients based on similar distribution, after which the clusters are passed to the global model for training. A proxy model is used to rank the clients based on performance. Only the weights of the best performing clients are used for training of the global model.

Experimental results, conducted using the CIFAR-10 dataset, demonstrate that FedNIP outperforms FedAvg, the most established FL algorithm, and matches FedProx, the most established Non-IID FL algorithm, in highly heterogeneous environments. Scaling the number of clients from 50 to 250 does not change the results. Hereby, using the FedNIP strategy of only using a subset of the clients (using top 10% performing clients in a cluster and 10% of random clients in a cluster) has similar performance compared to using FedNIP where all the clients are utilized in a cluster. This outcome means that examining a subset of clients within a cluster provides a reliable indication of the overall performance of the entire cluster. Which drastically reduces the number of needed clients to be used. FedNIP's runtime is four to eight times faster than FedAvg and FedProx, dependent on number of clients and level of statistical heterogeneity.

Future research should focus on integrating cluster performance as a sampling criterion for each round, instead of the current client proportion-based sampling strategy. Moreover, assessing FedNIP's performance in real-time environments could provide valuable insights, given its capability to dynamically rank clients based on performance. The code for the implementation of FedNIP can be found here: https://github.com/Chessmaster97/_FedNIP_

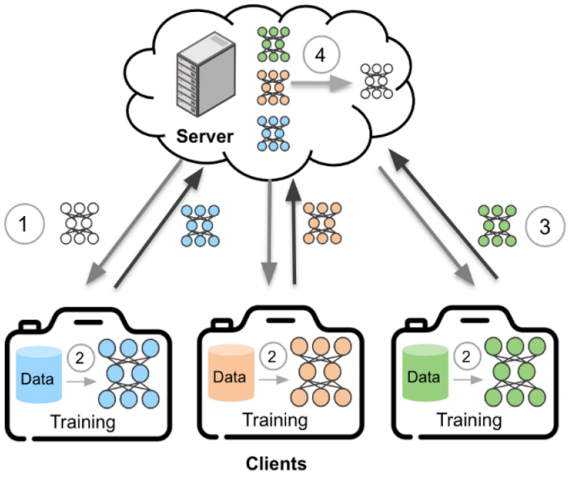

Figure 1: FL Training cycle [39].

1 INTRODUCTION

1.1 Context

In 2020, 2.5 quintillion bytes of data were created every day [24]. That is equivalent to 10 million Blu-ray discs, which when stacked would be as tall as two Eiffel Towers combined [6]. By 2025, the global data volume expands to 175 zettabytes. It is estimated that more than 50% of this data comes from IoT devices [24]. Furthermore, it is predicted that 150 trillion gigabytes of real-time data is analyzed [6]. The growth of this data over the last couple of years has been exponential, and as the software and hardware for storing, processing and analyzing gets more accessible, more organizations can benefit from this trend by incorporating strategies to support their business models.

For the analysis of this data, Machine Learning (ML) methods are more frequently used [5]. Especially, Deep Learning (DL) models thrive on a large amount of data and are increasingly used in modern data intensive applications. The use of ML has propelled several developments in areas of Natural Language Processing (ChatGPT), Computer Vision (Smart Cars) and Speech Processing (Alexa & Siri).

The classical way of using ML is by centralizing the data and executing the entire training process at a central server. In this way, pre-processing and fine-tuning, two important steps in ML, can perform extensively to enhance the model. However, a major drawback of this approach is that the data

is not private. It is often the case that a third party does the analysis of the data by the use of ML, and therefore all the raw data has to be delivered to this party. Even though contractual agreements prohibit the use of the data in any other way then for the purpose of ML, risk is still involved. Especially the healthcare sector, law enforcement and other governmental related organizations are hesitant to share this data [6]. Hence, potentially a lot of valuable insights are missed by organizations unwilling to share the raw data. Moreover, the new regulations, such as the GDPR, are also getting more strict about sharing and processing (sensitive) data [7].

Imagine a scenario where various hospitals across a region are collectively working to develop a predictive model for diabetes. Each hospital possesses its own set of patient data, comprising medical records, lab results, and other statistics. In a traditional centralized approach, all this sensitive patient data would need to be aggregated and sent to a central server for model training. With this, there is a major privacy and security concern.

Aggregating patient data from multiple hospitals into a central server increases the risk of privacy breaches. Patient data, including medical records, lab results, and other statistics, contains highly sensitive information that, if compromised, could lead to privacy violations for the individuals involved. Furthermore, centralizing sensitive data increases the risk of cyberattacks. A single point of failure, such as a central server, becomes a prime target for malicious attackers seeking to gain unauthorized access to valuable medical information. Without security measures in place, the data is vulnerable to unauthorized access, potentially compromising patient safety.

This is where FL steps in. Instead of sending the raw data to a central location, the hospitals can keep their patient data locally. The model is sent to the hospitals for local training, instead of bringing the data to the model. Google was the first to introduce the concept of FL as a possible solution for the challenges described. In 2016 Google published a paper titled Communication-Efficient Learning of Deep Networks from Decentralized Data [8]. Here, McMahan et al. introduces the concept of FL and the design of the federated averaging algorithm (FedAvg). FedAvg is the most commonly used algorithm used in FL. The common FL process is displayed in figure 1 [39]. The first step in FL is to share a model (composed at the server) with the available clients. Next, the clients that received the model can train their local data on this model (step 2). After the training is finished on the local model, a client sends the weights back to the server (step 3). The final step is to aggregate the weights received by the clients (step 4). Step 2 until step 4 are repeated until the global model has converged. After the publication of the mentioned paper, a lot of research has been conducted in a lot of different areas to examine the applicability of FL.

1.2 Problem description

This paper primarily addresses improving performance in case of Statistical Heterogeneity in data, also referred to as Non-Identically Independent Distributed Data (Non-IDD). In practical scenarios, clients possess diverse data distributions, and the data volume is often distributed in a scattered manner both locally and globally.

At a local level, there tends to be class imbalance, where certain classes may have more data compared to others. On a global level, the data distributions and class samples vary among different data clients.

In a study conducted by Jie Liu [11], a probability density function was generated for three clients using the Dirichlet distribution to show the suboptimal performance of the FedAVG algorithm. The visualization of this function can be found in figure 2.

In the figure, the performance of the regularly employed FedAvg algorithm is represented by a black cross, while the best performing region is indicated by a yellow star. It is apparent that relying solely on the FedAvg algorithm's performance yields suboptimal results. Consequently, the proposed FedNIP method strives to enhance performance by approaching the yellow star more closely.

Several other FL methods aim to approach the yellow star. In table 1 (page 10), the distinctions among FL algorithms addressing Non-IID scenarios, including FedNIP, are highlighted.

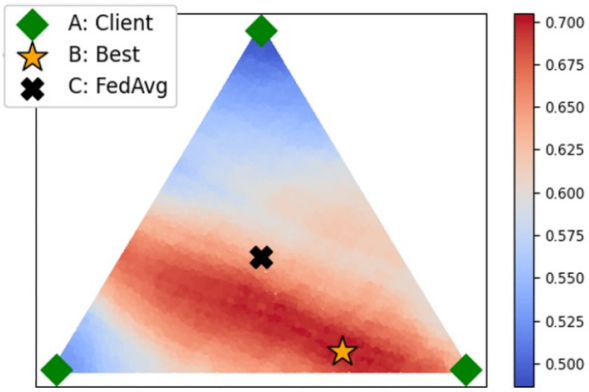

Figure 2: Dirichlet distribution with three clients indicating the sub-optimal performance of FedAvg [11].

1.3 Research Goal

The primary goal of FedNIP is to enhance both the global model's accuracy and speed, especially in scenarios involving statistical heterogeneity, surpassing the performance of baseline classifiers like FedAvg and FedProx. Among Federated Learning algorithms, FedAvg holds the status of being the most extensively utilized, whereas FedProx stands out as the established benchmark for Non-IDD data. While FedProx and FedNIP share the common objective of improving FL performance in heterogeneous environments, they have different approaches to achieve this goal. This is highlighted in table 1 on page 10.

The main research question is:

- How does the performance of FedNIP, in terms of global accuracy and speed, compare to that of baseline classifiers FedAvg and FedProx for clients with statistical heterogeneous data?

The following sub questions are answered:

- To what extent does FedNIP influence the accuracy of the global model compared to baseline classifiers?
- What is the influence of scaling the number of clients on the performance of FedNIP?
- What are the effects of FedNIP under different non-identically distributed (Non-IDD) settings using the Dirichlet distribution?
- To what extent does FedNIP affect the speed compared to the baseline methods?

2 BACKGROUND

2.1 Deep Learning

As the amount of data increased exponentially and hardware capabilities became more accessible to the wider public, the utilization of Deep Learning (DL) models has witnessed significant growth [18]. This increase in popularity can be attributed to the superior performance of Deep Learning models compared to traditional machine learning (ML) algorithms such as Decision Trees, Support Vector Machines, Naive Bayes, and Logistic Regression, particularly when dealing with large datasets [16]. DL models excel in capturing complex nonlinear relationships, surpassing the capabilities of traditional algorithms and yielding higher performance [3].

The effectiveness of DL has made it the state-of-the-art approach for various ML applications across different use cases [16]. Neural networks, a class of models inspired by the structure and functionality of the human brain, form the foundation of DL. These networks consist of interconnected neurons that process information and generate predictions based on input data. Among the most prevalent types of neural networks used in deep learning is the feedforward neural network, characterized by layers of neurons responsible for processing input data and producing output predictions [21]. The input layer receives raw data, and each subsequent layer performs nonlinear transformations on the outputs of the previous layer. Finally, the output layer provides predictions based on the given input [3].

Let $x \in \mathbb{R}^n$ be the input vector, and $y \in \mathbb{R}^m$ be the output vector. A feedforward neural network can be represented as a function $f(x;\theta)$, where θ denotes the parameters of the network. The network computes its output by propagating the input vector through the layers:

$$y = f(x;\theta) = f_L(\dots f_2(f_1(x;\theta_1);\theta_2)\dots;\theta_L) \tag{1}$$

Here, $f_1, \dots, f_L$ denote the nonlinear activation functions of the layers, and $\theta_1, \dots, \theta_L$ are the corresponding parameters.

To train a neural network, a loss function can be defined that measures the discrepancy between the predicted output and the true output. The most common loss function used in deep learning is the cross-entropy loss [16]:

$$\mathcal{L}(\theta) = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{m} y_{ij}\log f(x_i;\theta)_j \tag{2}$$

$$+ (1-y_{ij})\log\big(1-f(x_i;\theta)_j\big) \tag{3}$$

Here, N is the number of training examples, y_{ij} is the j-th component of the true output for the i-th example, and $f(x_i;\theta)_j$ is the j-th component of the predicted output for the i-th example.

To minimize the loss function, we use an optimization algorithm, such as stochastic gradient descent (SGD), to update the parameters of the network [3]:

$$\theta \leftarrow \theta - \alpha\nabla_\theta\mathcal{L}(\theta) \tag{4}$$

Here, α is the learning rate, and $\nabla_\theta\mathcal{L}(\theta)$ is the gradient of the loss function with respect to the parameters θ.

Using back propagation it is possible to efficiently computes the gradients by propagating the errors from the output layer back to the input layer. The back propagation algorithm works by computing the derivative of the loss function with respect to the output of each layer, and then recursively

applying the chain rule to compute the derivative of the loss function with respect to the parameters.

Let $\delta_i^{(l)}$ denote the error term for the i-th neuron in layer l, which is defined as the derivative of the loss function with respect to the output of that neuron. The backpropagation algorithm computes the error terms for each layer as follows [16]:

$$\delta^{(L)} = \nabla_{\hat{y}} \mathcal{L} \odot f_L'(\hat{y})\ \delta^{(l)} \quad = (W^{(l+1)})^T \delta^{(l+1)} \odot f_l'(z^{(l)}) \tag{5}$$

Here, $\hat{y}$ is the predicted output, $\odot$ denotes element-wise multiplication, $W^{(l)}$ is the weight matrix for layer l, $f_l'(z^{(l)})$ is the derivative of the activation function for layer l, and $z^{(l)}$ is the weighted sum of the inputs to layer l.

Once the error terms are computed, the gradients of the loss function with respect to the parameters can be computed using the chain rule:

$$\nabla_{W^{(l)}} \mathcal{L} = \delta^{(l+1)} (a^{(l)})^T\ \nabla_{b^{(l)}} \mathcal{L} \qquad = \delta^{(l+1)} \tag{6}$$

Here, $a^{(l)}$ is the output of layer l after applying the activation function, and $b^{(l)}$ is the bias vector for layer l.

Deep learning models are prone to overfitting, where the model memorizes the training data instead of learning generalizable patterns. To address this issue, various regularization techniques can be applied to the model.

One common technique is weight decay, where a penalty term is added to the loss function to discourage large weights:

$$\mathcal{L}(\theta) = \mathcal{L}\text{data}(\theta) + \lambda \sum l = 1^L ||W^{(l)}||_2^2 \tag{7}$$

Here, $\mathcal{L}_{\text{data}}(\theta)$ is the data loss, $||W^{(l)}||_2$ is the L_2 norm of the weight matrix for layer l, and λ is the weight decay coefficient.

Another technique is dropout, where a random subset of neurons are temporarily removed from the network during training to prevent co-adaptation:

$$a'^{(l)} = r^{(l)} \odot a^{(l)} \tag{8}$$

Here, $r^{(l)}$ is a binary mask that randomly sets some of the activations to zero.

2.2 Federated Learning

The objective of the Federated Learning (FL) framework in this setup can be written as [30]:

$$\min_W F(W) = \frac{1}{|D|} \sum_{i=1}^{N} |D_i| F_i(W), \tag{9}$$

where $|D|$ is the size of the global dataset D and $|D_i|$ is the size of the local dataset D_i in client i. $F(W)$ is the global cost function on D and $F_i(W)$ is the local cost function on D_i. The loss function of data samples is denoted by L.

This paper considers the Federated Averaging (FedAvg) algorithm [20], which executes the following three steps every r-th iteration:

(i) Initialization: All clients receive the global model $\bar{w}r-1, \bar{S}r-1$ in the last iteration from the server.

(ii) Local model updating: Each client $i \in [N]$ updates its local gradient parameters through E successive steps of local gradient descent. Specifically, each client initializes the local model by

$$w_{r,0}^{(i)}, \bar{S}r, 0^{D_i} = \bar{w}r-1, \bar{S}_{r-1}, \tag{10}$$

and performs the following update for $t \in [E]$:

$$w_{r,t}^{(i)} = w_{r,t-1}^{(i)} - \gamma \nabla_w F_i(w_{r,t-1}^{(i)}; \bar{S}r, t^{D_i}, \Delta\bar{S}r, t^{D_i}), \tag{11}$$

where $\gamma > 0$ is the learning rate, E is the number of local updating steps, $\bar{S}r, t^{D_i}$ is the statistical parameters computed via (2) with the local model $wr, t-1^{(i)}$ and the local dataset D_i, and $\Delta\bar{S}_{r,t}^{D_i}$ is the difference between the local and global statistical parameters. During this time, each client updates local statistical parameters for model inference by moving average as follows:

$$\bar{S}r, t^{D_i} = (1-\rho)\bar{S}r, t-1^{D_i} + \rho\bar{S}_{r,t}^{D_i},\ t \in [E], \tag{12}$$

where ρ is the momentum of moving average.

(iii) Aggregation: The local model $w_{r,E}^{(i)}, \bar{S}_{r,E}^{D_i}$ in each client $i \in [N]$ is uploaded to the server for aggregating a new global model by

$$\bar{w}r, \bar{S}r = \sum i = 1^N p_i wr, E^{(i)}, \bar{S}_{r,E}^{D_i}, \tag{13}$$

where $p_i = |D_i|/|D|$ is the weight assigned to client i.

2.3 Flower

Flower is an upcoming Federated Learning Framework that enables developers to create research projects that can be transformed into production deployment [2]. The framework is developed by the German Startup Adap. It is designed to be agnostic, which means that it supports several ML frameworks like TensorFlow and PyTorch, and it is programming language independent (Python, Java, C++ ...). The design makes the scaling of the number of clients manageable. Supported are execution on GPUs as well as containerization using tools like Docker. Still, a lot of functionalities need to be added to the Framework. Differential Privacy and Secure Aggregation are not yet implemented in the Framework.

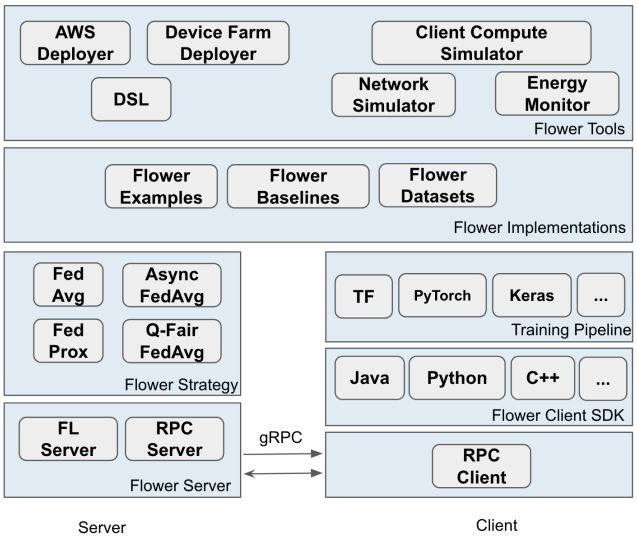

Figure 3: Flower Framework [2]

In figure 5 the Flower architecture is visualized. The flower architecture can be deployed using several platforms. For example, the AWS Deployer, Device Farm Deployer, the Client Compute Simulator and the Network Simulator. The Flower framework also provides several baselines of FL that can be used. Flower supports multiple languages. For this research, Pytorch is used as AI framework.

For this research, a new algorithm (FedNIP) is developed. The Flower framework provides the option to create a custom strategy for this purpose. The strategy abstraction enables implementation of fully custom strategies [1]. A strategy is the federated learning algorithm that runs on the server. Strategies decide how to sample clients, how to configure clients for training, how to aggregate updates, and how to evaluate models.

[1]https://flower.ai/

2.4 Evaluation Metrics

The global accuracy and convergence speed are used as main metrics to compare algorithms. The global accuracy for clients is defined as follows:

Let n be the total number of clients in the federated learning system, and let m_i be the number of samples in the local dataset of client i, where $i \in [1, n]$. Let y_{ij} be the ground truth label of the j-th sample in the local dataset of client i, and let $\hat{y}_{ij}$ be the predicted label for this sample.

Suppose that in previous rounds, we have selected a subset S_k of clients to participate in training, where $k \in [0, 100]$, and let c_k be the best performing client among them. In addition, we randomly select another subset R_k of $k\%$ clients to participate in the training, and let r_k be the best performing client among them. We merge the two subsets to obtain the final set of clients for the round: $F_k = S_k \cup R_k$.

In the current round, we simulate one round of training using only the local models of the clients in F_k, and let $\hat{w}_k$ be the resulting global model.

We then update the global model by training it on the local datasets of all clients, but using only the weights of c_k as the starting point. The resulting model is denoted by $\hat{w}$.

If the best performing client among the F_k clients is significantly different with a threshold that is empirically determined from c_k, then we swap the rankings of the clients in the two subsets S_k and R_k, and repeat the training process as described above.

With regard to the convergence speed, let's assume we have a sequence of accuracy values over successive training rounds, denoted as $A = \{A_0, A_1, A_2, \ldots, A_t\}$, where A_t represents the accuracy at training round t.

To determine convergence we can define a convergence criterion using a threshold ε. We observe the accuracy values and check if the accuracy remains relatively stable over a certain period of time. If the accuracy does not improve significantly beyond ε for a predefined number of rounds, we can assume convergence.

The convergence criterion can be defined as follows: If there exists a positive integer n such that for all $t > n$, $|A_t - A_{t-n}| < \varepsilon$, where $|A_t - A_{t-n}|$ represents the absolute difference between the accuracy at round t and the accuracy at round $t-n$. The condition $|A_t - A_{t-n}| < \varepsilon$ implies that the accuracy values are within a small threshold ε, indicating stability. In this criterion, the parameter n determines the number of rounds over which the accuracy must remain stable.

The Dirchlet distribution is used to control the degree of Non-IDD in the used datasets. The Dirichlet distribution is a

multivariate probability distribution that assigns probabilities to vectors of non-negative numbers that sum to one [14]. It is commonly used as a prior distribution in Bayesian statistics, particularly in problems that involve estimating probabilities of events with a categorical or multinomial structure.

More formally, let $\mathbf{x} = (x_1, x_2, \ldots, x_K)$ be a vector of K non-negative values that sum to one, and let $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \ldots, \alpha_K)$ be a vector of positive hyperparameters. Then the probability density function of the Dirichlet distribution is given by:

$$f(\mathbf{x}; \boldsymbol{\alpha}) = \frac{1}{B(\boldsymbol{\alpha})} \prod_{i=1}^{K} x_i^{\alpha_i - 1}, \tag{14}$$

where $B(\boldsymbol{\alpha})$ is the multivariate Beta function, which serves as a normalization constant to ensure that the probabilities sum to one [32]. The Beta function is defined as:

$$B(\boldsymbol{\alpha}) = \frac{\prod_{i=1}^{K} \Gamma(\alpha_i)}{\Gamma(\sum_{i=1}^{K} \alpha_i)}, \tag{15}$$

where $\Gamma(\cdot)$ is the Gamma function.

The hyperparameters $\boldsymbol{\alpha}$ determine the shape of the distribution and can be used to encode prior beliefs about the probabilities of events.

In this research, experiments are done with values of $\boldsymbol{\alpha}$ = 0.6 ,alpha = 0.3 and alpha of 0.05 as this is common in other research as well [12][5][4]. The lower the level of alpha the higher the degree of statistical heterogeneous data. By experimenting with these three values (0.6,0.3 and 0.05) it is possible to test different algorithms for low, moderate and high degrees of statistical heterogeneous data.

The Earth Mover's Distance (EMD) is used to compare local data distributions [28]. It is a measure of the dissimilarity between two probability distributions over a certain space. Given two probability distributions $P = p_1, p_2, \ldots, p_n$ and $Q = q_1, q_2, \ldots, q_m$, defined over a space X, the EMD is defined as the minimum amount of work required to transform one distribution into the other.

More formally, let $d(x, y)$ be the distance between two points x and y in X. Then the EMD between P and Q is defined as:

$$\mathrm{EMD}(P, Q) = \min_{f} \sum_{i=1}^{n} \sum_{j=1}^{m} f_{ij} \cdot d(x_i, y_j), \tag{16}$$

where f_{ij} represents the amount of probability mass to be moved from p_i to q_j, subject to the following constraints:

$$\begin{aligned} &\sum_{j=1}^{m} f_{ij} \leq p_i, && \forall i \in [1, n], \\ &\sum_{i=1}^{n} f_{ij} \leq q_j, && \forall j \in [1, m], \\ &f_{ij} \geq 0, && \forall i \in [1, n], j \in [1, m]. \end{aligned} \tag{17}$$

3 RELATED WORK

There have been several studies addressing the issue of statistical heterogeneity in FL. Zhu [34] conducted a comprehensive survey to explore the strategies employed by researchers in tackling this challenge. The survey revealed that researchers primarily concentrate on four approaches: algorithmic-based, model-based, data-based, and framework-based approaches.

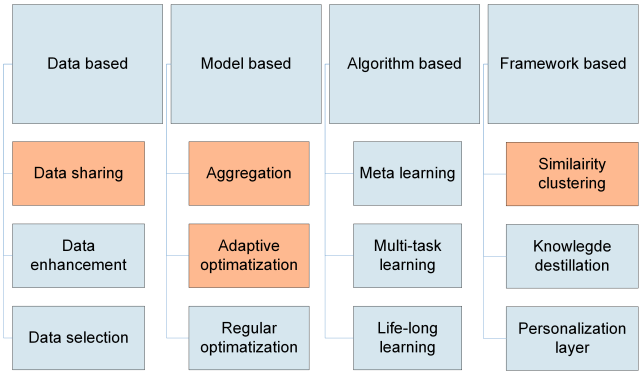

Figure 4: Strategies of reseachers in FL in case of Non-IDD FL [34].

In this study, four components of the model (see figure 4) are employed to handle statistical heterogeneous data: data sharing, aggregation, adaptive optimization, and similarity clustering. Building upon these components, several algorithms that incorporate similar elements are explored and discussed below.

FedAvg [8] averages the weights of the local models at the server after each communication round. Hereby, clients are selected randomly. Several studies prove that FedAvg suffers from performance issues when the data is statistical heterogeneous in nature.

FedProx [27], a widely adopted FL algorithm, adds a proximal term (which can be tuned with parameter μ) to limit the distance between the local and global model. In practise, it is difficult to find an optimal value for μ, if μ is too small then the proximal term has a negligible effect (same performance as FedAvg) and if μ is too large then convergence speed is slowed down as the local updates are small. FedProx is tested in highly heterogeneous settings, FedProx demonstrates significantly more stable and accurate convergence behavior relative to FedAvg—improving absolute test accuracy by 22% on average.

FedAdam [23] uses a adaptive learning rate where clients perform multiple epochs of training using a client optimizer to minimize loss on their local data and the server updates its global model by applying a gradient-based server optimizer to the average of the clients' model updates. Emprical analysis shows that selection of client and server learning rates can reduce the effect of client heterogeneity. However, it does not completely remove it. In highly heterogeneous settings, FedAdam is not performing stable.

FedCS [26] aggregates more updates by selecting more clients with less resource constraints in one round, it offers robustness against straggler devices by allowing the remaining devices to continue local optimization without waiting for the straggler devices to finish. They deal with the imbalance of the data by using a compression technique, that could increase the number of clients that can be selected in the same communication round. The results indicate that the overall performance were limited in case of Non-IDD environments. They achieve a maximum accuracy of 70%.

Class-aware Client Selection (C2S) [17] clusters the clients according to the classes of data they have. A cluster is selected in every communication round to participate for the training on the global model. Here, the clients in a cluster are averaged in order to create a set of weights for the global model. For fairness, the probability for each client getting selected is equal. Furthermore, they take make sure that clients that have participated less in training compared to other clients by introducing a value set system. If a set has the minimum value, the set gets selected for training. For CIFAR-10, FedC2S has more obvious accuracy improvements over FedAvg and FedCluster, which are 13.12% and 14.36%, respectively. It also has significant reduction in number of rounds to achieve accuracy of 73%.

FedCluster [4] allows clusters on devices on different type of application scenario's. Random uniform clustering groups devices of equal size in a cluster at random. Moreover, timezone-based clustering allows clustering based on timezone or GPS location. Lastly, availability-based clustering is based on the idea to divide each learning round into multiple time slots. The available devices within a cluster can form a cluster. With regard to Non-IDD data, FedCluster performs worse than the FedAvg algorithm.

BalanceFL [35] addresses class imbalance by long tail FL. They designed a novel local update scheme that adjusts the class imbalance, forcing the local model to behave as if it were trained on ideal uniform distributed data. Hereby, three steps are used. Firstly, they used balanced sampling to increase the probability of data from tail classes to be chosen to balance the response of all classes during the training. Secondly, feature-level data augmentation is used to increase the samples for the minority classes. Lastly, the smooth regularization penalizes the over-confident predictions for better balancing and representation learning. Lastly, smooth regularization penalizes the over-confident predictions for better

balancing and representation learning. They tested their approach a real-life IMU dataset for action recognition, we collect a real-life IMU dataset for action recognition, which includes over 10,000 data samples. For the evaluation, they used three datasets from three different data modalities. The results show that under all datasets, BalanceFL performs significantly better than others. Specifically, on the long-tailed version of CIFAR10, BalanceFL outperforms the FedAvg by up to 56.7% in terms of accuracy, while incurring 75% less communication overhead.

Auto-FedAvg [36] makes use of aggregation weights which are dynamically adjusted, depending on data distributions across data silos and the current training progress of the models. Hereby, gradient descent is used with the Dirichlet distribution in order to capture the underlying data and learning process. Auto-FedAvg outperforms FedAvg in both limiting the convergence speed as increasing the accuracy.

RepFL [31] is designed to measure the reputation of each of the clients and based on this reputation, clients can participate in the training process. The reputation is determined based on the local accuracy of each of the clients, by using a copy of the global model and by using the performance of the previous round. They make use of weighted aggregation to give the better performing clients more weight. Hereby, they make use of the normal distribution and divide regions based on the standard deviation. The algorithm improves both the accuracy (improvement of 17.175%) and the convergence speed.

FedDC [13] makes minimal modifications to the local training phase, where each client uses a lightweight auxiliary local drift variable to track the difference between its local model parameters and the global model parameters. The aim of FedDC is to leverage this learned local drift variable to align the parameters, promoting consistency at the parameter level. Empirical results and analysis show that FedDC achieves faster convergence and improved performance on image classification tasks, and is robust to partial participation, non-i.i.d. data, and heterogeneous clients.

TiFl [37] utilizes a tier-based approach to categorize clients according to their training performance. This method selects clients from the same tier during each training round, aiming to address the straggler issue that arises due to resource and data quantity heterogeneity. TiFL goes a step further to tackle the non-IID data and resource heterogeneity by adopting an adaptive tier selection technique. This approach updates the tiering dynamically, based on the observed training performance and accuracy.

FedNova [10] is a technique that accurately normalizes the updates made by local models during averaging. The central concept of FedNova involves computing the average of normalized local gradients.

$$\frac{x(t,\tau_i)_i - x(t,0)}{\tau_i}$$

Where $x(t,\tau_i)_i$ represents the cumulative local gradient returned by client i after performing τ_i local updates in the t-th training round. This approach replaces the method of averaging the cumulative local gradient, which is represented using the formula:

$$x(t,\tau_i)_i - x(t,0)$$

Scaffold [22] adds a control variate regularization term that induces variance to correct for the client-drift in local updates. Hereby, the federated training process is divided into two phases: a scaffolding phase and a fine-tuning phase. In the scaffolding phase, the algorithm first trains a shared model on a representative subset of the data from a subset of clients. This representative subset of data, also known as scaffold data, is chosen such that it covers the diversity of the data distributions among all clients. In the fine-tuning phase, each client fine-tunes the shared model on its own local data to further improve the performance.

ABAVG [9] incorporates a weighted averaging system that relies on the accuracy achieved in the validation set. However, this necessitates sharing some data with the server before the start of Federated Learning (FL). The validation set is utilized to compare the accuracy's attained by multiple clients, and based on this assessment, appropriate weights are assigned to each client. In addition, clustering based on data distribution is implemented to ensure equitable comparisons between clients. They train each cluster on a seperate global model. In various data distributions, the implementation of the ABAVG algorithm led to an average increase in convergence speed of 47%in the Mnist dataset, 59% in the Mnist dataset, and 33% in the CIFAR-10 dataset.

ACSFed [1] is an algorithm that uses a probabilistic approach to client selection. They also use the EMD to cluster the clients and create a probability matrix where clients get assigned a certain probability of selection. Hereby, they make sure that clients that have a high degree of Non-IDD get sampled more often in order to create fairness. Hereby, they also assign a higher probability of clients that have not been selected often. The algorithm is compared to the standard FedAvg algorithm and obtains a higher performance then FedAvg

FAug [7] uses generative adversarial networks to make the local device datasets closer to an independently and identically distributed distribution. The clients perform data augmentation on their local data and send the augmented data to

a central server for training a shared model. This process continues until a desired accuracy is achieved or a predetermined stopping criterion is met. The use of data augmentation in FAug enhances the diversity of the data seen by the shared model, leading to improved performance. Moreover, by artificially expanding the size of local datasets, data augmentation also addresses the challenge of limited data availability on each client. Empirical studies have shown that FAug is effective in achieving high accuracy with reduced communication overhead compared to traditional FL algorithms.
(t,k) = Retrain(fk, E, L(t-1,median)) if L(t,k) > L(t-1,median), and w(t,k) = w(t,0,k) otherwise.

FedCD [12] is a new framework for federated learning that is based on knowledge distillation and collaborative learning. This framework allows users to design models independently, while also introducing a communication-efficient scheme for sharing local model updates with the parameter server. This scheme is based on online knowledge distillation. Overall, the proposed approach aims to improve the efficiency and effectiveness of federated learning.

CSFedAvg [33] uses the observation that clients with Non-IID data exhibit varying weight divergence compared to those with IID data. Based on this observation, weight divergence is used to identify the Non-IID degree of clients. Clients with lower Non-IID degrees are chosen more frequently, resulting in a more efficient FL process.

Astrea [19] addresses data imbalance issues in federated learning, using two methods: global data distribution based data augmentation, and mediator based multi-client rescheduling. Astraea reduces global imbalance through runtime data augmentation and resolves local imbalance by creating a mediator to reschedule client training based on Kullback-Leibler divergence (KLD) of their data distribution. Compared to FedAvg, Astraea shows significant improvement of top-1 accuracy on the imbalanced EMNIST and CINIC-10 datasets, with lower communication traffic. Specifically, Astraea shows +5.59% and +5.89% improvement on these datasets, respectively, while reducing communication traffic by 82%.

AdaFed [38] can dynamically adjust the aggregation weight of each device by considering its historical participation records, thus addressing the bias caused by partial device participation. Empirical results validate the theoretical analysis, demonstrating that AdaFed significantly enhances the accuracy of the global model and achieves faster convergence compared to existing FL methods by mitigating the negative impact of biased device participation.

FedFa [25] aligns the feature mappings and calibrates classifiers across clients by updating client models in a shared feature space with consistent classifiers. This leads to a virtuous cycle between feature consistency and classifier similarity across clients. Experiments demonstrate that FedFA outperforms federated learning algorithms to image classification datasets with label and feature distribution skews.

FedBS [5] is a novel algorithm to handle global models having batch normalization layers, in the presence of Non-IID data. FedBS modifies FedAvg by introducing a new aggregation rule at the server-side, while also retaining full compatibility with Batch Normalization (BN).

ratioloss [15] is an algorithm that monitors the composition of training data round by round. When detecting a similar imbalanced composition continuously, the system acknowledges the class imbalance and load the Ratio Loss. This holds true for both local and global imbalances. It adjusts the loss function accordingly.

HybridFL [20] is a new hybrid Federated Learning (FL) strategy that can be customized to fit various FL settings. This model addresses the key requirements of collaborative-learning scenarios, where neither the subject nor feature sets are complete at any client. The authors also develop a convergent hybrid FL algorithm that allows for knowledge transfer between clients, while maintaining data locality and improving communication efficiency by removing the need for sample synchronization. The performance of this new model was evaluated on a real dataset, and the results showed that the learned model achieved comparable accuracy to a centrally trained model.

Table 1: Comparison of FL Algorithms dealing with Non-IDD

Reference	FL Algorithm	Client Priority	Cluster Based	Dynamic	Method	Raw Data Exposure
[8]	FedAvg	X	X	X	Select Clients Random	X
[27]	FedProx	X	X	X	Adding Proximal Term	X
[23]	FedAdam	X	X	X	Adaptive Learning Rate	X
[26]	FedCS	✓	X	X	Compression Technique	X
[17]	C2S	X	✓	X	Average of Clients in Same Cluster	X
[4]	FedCluster	X	✓	X	Allows Different Types of Clustering	X
[35]	BalanceFL	X	✓	X	Balanced Sampling and Smoothing	X
[36]	Auto-FedAvg	✓	X	✓	Dynamically Adjusted Weights	X
[31]	RepFL	✓	X	X	Based on Reputation Client Assign Weight	X
[13]	FedDC	X	X	X	Lightweight Auxiliary Local Drift Variable	X
[37]	TiFL	✓	✓	X	Adaptive Tier Selection Algorithm	X
[10]	FedNova	X	X	X	A Normalized Averaging Method	X
[22]	Scaffold	X	X	X	Variance Reduction for Client-Drift	X
[9]	ABAVG	✓	✓	X	Weighted Accuracy System	✓
[1]	ACSFed	✓	✓	X	Probability Matrix based on Client Importance	X
[7]	Faug	X	X	X	Using Generative Adversarial Networks	X
[12]	FedCD	X	✓	X	Dynamically Group Devices with Similar Data	X
[33]	CSFedAvg	X	✓	X	Non-IDD Clients are Sampled Less Often	✓
[19]	Astraea	X	X	X	Global Data Augmentation and Rescheduling	X
[38]	AdaFed	✓	X	X	Adaptively Tune the Aggregation Weight	X
[25]	FedFA	X	X	X	Feature Anchors to Align Feature Mappings	X
[5]	FedBS	✓	X	X	Global Models Having Batch Normalization Layers	X
[15]	Ratio Loss	X	✓	X	Designing New Loss Function	X
[20]	Hybrid-FL	X	X	X	Model-Matching-Based System	X
This work	FedNIP	✓	✓	✓	Dynamic Ranking Based System	X

Table 1 presents a comparison of the discussed algorithms. It evaluates them based on whether they prioritize certain clients, utilize clustering to address Non-IID data, their primary method, and whether they expose raw data to the server, such as a validation set.

This table highlights Auto-FedAvg, TiFl, RepFL, ABAVG, and ACSFed as the algorithms most closely resembling FedNIP. FedNIP is different from other FL algorithms in the sense that it dynamically updates training performance based on ranking, where only the best clients get selected for training. Most other FL algorithms either use a validation set or make use of fixed weights.

Only Auto-FedAvg sets the weights dynamically based on difference in data distribution and based on performance. Weights are dynamically adjusted based on client performance and distribution. FedNIP does not focus on weights adjustments. It makes use of clustering and then uses a proxy model to rank clients. In comparison to Auto-FedAvg, which involves all the clients in the training process, FedNIP offers the flexibility to utilize only a subset of clients. By focusing the most influential clients, FedNIP potentially achieves faster training times while maintaining model performance.

ABAVG uses a validation set to set the weights of the clients. This leaks raw data to the server, which is not ideal. Moreover, fixed weights are used for the entire training process, which are based on this validation. This could lead to long term bias towards clients that happen to perform better in the validation set. The validation set is a small portion of the entire training set and might not present a representative overview, causing the model to drift in the wrong direction. FedNIP makes use of a warm-up period that does not expose the raw data and dynamically changes ranking since performance of the clients might change over time.

RepFL uses several metrics to indicate the reputation of a client. Based on this, they assign higher weights to better performing clients. They do not cluster the clients based on data distribution, hence it is likely that idd clients perform better and therefore is weighted to heavenly. Neglecting the inclusion of Non-IDD data for a better generalization of the model. Furthermore, the authors state that the weights are set empirically and are not dynamically updated during the training, which could result in a long-term bias. ACSFed is an algorithm that uses a probabilistic approach to client selection. While both ACSFed and FedNIP make use of the same clustering technique and use an adaptive approach, there are

differences in the design of both algorithms. The key difference between both algorithms is that FedNIP aims to prioritize impactful clients, it does not explicitly focus on fairness in the selection process. The selection is primarily driven by the potential impact on the global model, rather than explicitly considering fairness or underrepresented clients. ACSFed explicitly considers fairness by assigning higher probabilities to clients that have not been selected often. This adaptive probability assignment ensures that underrepresented clients have a higher chance of participation, promoting fairness in the selection process. FedNIP does not involve explicit probability assignment to clients. The selection is based on rankings, and only the weights of the best-performing model are used for actual model training.

TiFL prioritizes clients based on speed and incorporates an adaptive tier selection mechanism to consider accuracy as well. By adjusting the probability, TiFL aims to achieve a more optimal balance between speed and accuracy. However, unlike FedNIP, TiFL does not group clients based on their data distribution. Instead, TiFL relies solely on past performance metrics when selecting clients. It does not take into account potential future contributions, as FedNIP does through the use of a proxy model.

4 EXPERIMENTAL DESIGN

In section 6 Results and Analysis, the effectiveness of the FedNIP algorithm is examined. Specifically, the global accuracy, runtime, scalability, performance on varying degrees of statistical heterogeneous data and the time it takes to convergence is investigated. This is compared with the baseline algorithms FedAvg and FedProx. In this section 4, the design of these experiments are described.

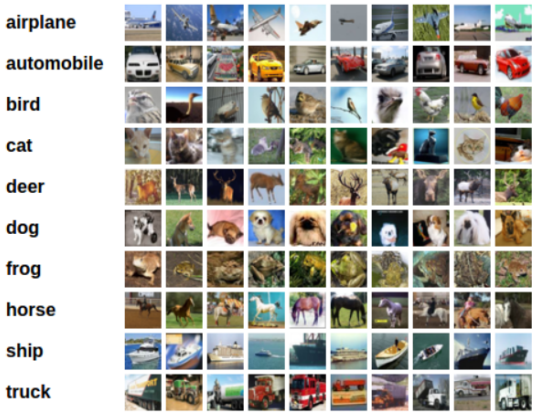

Figure 5: CIFAR 10 Dataset [29]

4.1 Dataset and Baselines

The dataset CFIR-10 [29] is used for the application of the FedNIP algorithm (see figure 5). As this dataset is commonly used among researchers [27, 23, 26, 22, 10, 25, 5] to test their developed FL algorithm. CIFAR-10 (Canadian Institute For Advanced Research) contains 60,000 32x32 color images in 10 different classes. The 10 different classes represent airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks. There are 6,000 images of each class. This dataset is split using the Dirchelet distribution depending on the number of clients used in the experiment. In an idd setting, training samples are randomly selected and equally assigned to clients. All the clients have the same amount of training data, and each client's data points are evenly distributed in all categories. In case of Non-IDD data, the label ratios follow the Dirichlet distribution.

4.2 Hyper-parameter Settings

The standard Fl architecture is used as indicated in figure 1. The local training epochs used is only 1 in each round, with a learning rate of 0.1. For the FedProx algorithm, $\mu = 10^4$ is used, as this is common in other works as well. Furthermore, the values of p and r are set to 0.1, indicating that 10% of the top-performing clients in a cluster are chosen (p), and 10% of random clients are chosen (r). All these parameters can be adjusted as explained in the README file that is published on GitHub.

The CNN has six layers: two convolutional layers (conv1 and conv2), two fully connected layers (fc1 and fc2), and one output layer (fc3). Initially, conv1 takes a 3×32×32 input (a color image with 3 color channels of size 32×32) and applies 6 filters of size 5×5, resulting in a 6×28×28 output. This output undergoes max pooling (pool) of size 2×2 with a stride of 2, yielding a 6×14×14 output. Subsequently, conv2 applies 16 filters of size 5×5 to this output, generating a 16×10×10 output. After flattening, the output passes through two fully connected layers (fc1 and fc2) of sizes 120 and 84, respectively, before reaching the final output layer (fc3) of size 10, representing the 10 possible classes for classification.

4.3 Implementation

As mentioned in section 2.3, the Flower framework is used for implementation[2]. Adjustments are made to certain methods within the Flower framework itself. Since Flower is still evolving, it hasn't reached a stage where every customization can be instantly applied through the API. The details of these changes can be found on GitHub.

[2]https://flower.ai/docs/framework/how-to-use-strategies.html

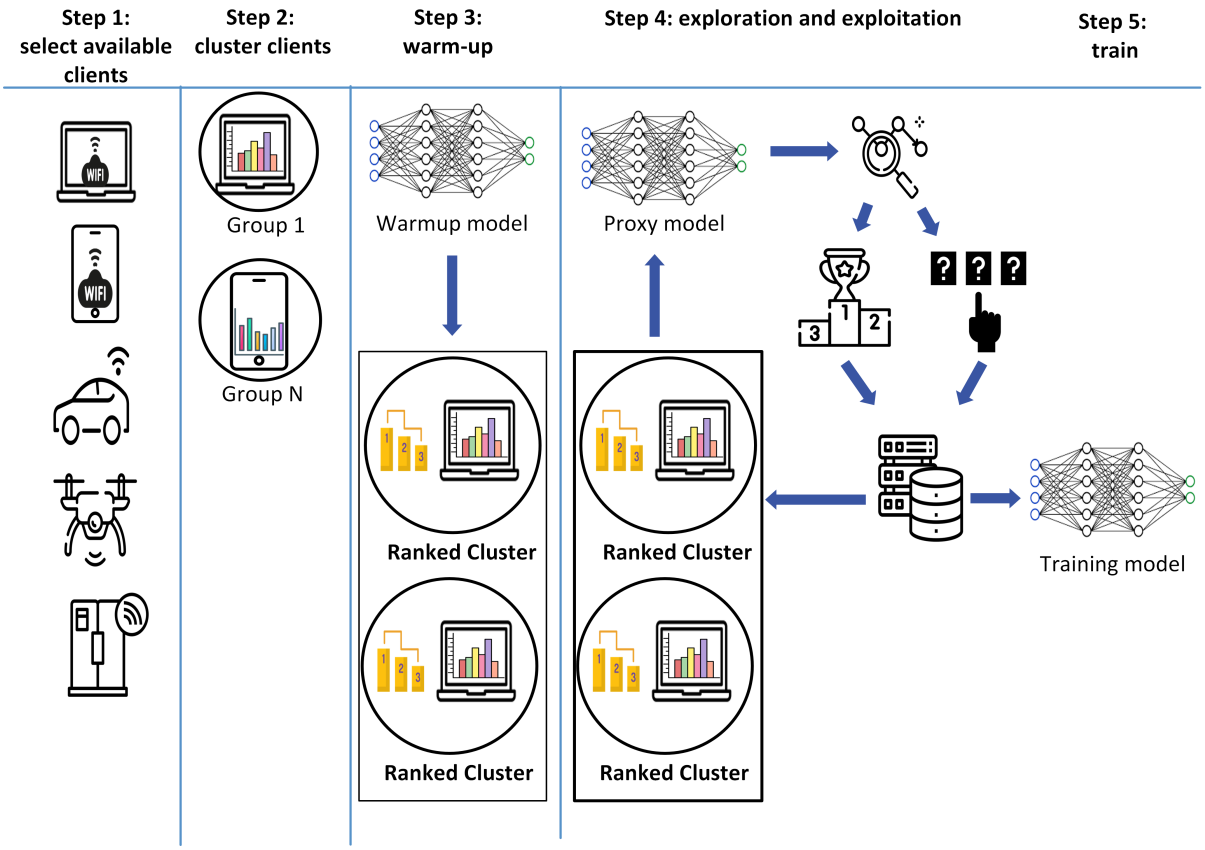

Figure 6: Design of FedNIP

5 METHOD

The design of FedNIP is visualized in figure 6. The process begins by selecting available clients for the next communication round (step 1). These clients are then clustered based on their similar local data distribution, employing the Earth Movers Distance (EMD) for a fair accuracy comparison. The clusters are calculated using K-Means clustering, with the EMD distances between client data distributions as the basis for determining similarity. The optimal number of clusters is determined using the Elbow Method, and then clients are assigned to clusters based on their similarities in data distribution. In round 1 the clients don't train, but sent their data distribution to the central server.

After which the server computes the EMD for the clients and assigns the clients in a cluster (step 2). Once the clustering is completed, the warm-up period is started (step 3). During this phase, the goal is to establish an initial ranking within each client cluster based on the warm-up period. Once the warm-up period concludes, each cluster has its clients ranked accordingly. The subsequent step involves initiating the model training. The selection of a client for training is based on their previous performance. Initially, this performance is determined using information from the warm-up period. As training progresses, the selection is based on the actual training performance.

Sampling takes place based on proportion of clients in a cluster. Hence, a cluster with more clients gets a higher chance of getting selected in a given communication round. For each training round, the top k% of clients, along with a randomly selected r% of clients, are utilized to simulate the next round. This simulation involves the use of a proxy model (step 4), which is a temporary copy of the global model, with different clients contributing. The performance of this round is stored, and the ranking is dynamically updated if a random client scores better than a current top performing client. This proxy model can be configured for certain rounds and does not need to be used every communication round. After this process, only the best weights are selected for training of the actual global model (step 5). In case a randomly selected client performs significantly better than a higher-ranked client, they swap positions in the ranking. This process continues until convergence of the global model is achieved.

6 RESULTS AND ANALYSIS

A series of experiments have been conducted to assess the effectiveness of FedNIP compared to FedAvg and FedProx. The convergence curves under different alpha levels of the Dirichlet distribution are studied. Hereby, we experiment with two variations of FedNIP. One variation is the FedNIP full, which means that every client in the cluster gets trained on the proxy model and the best weights of the client in the cluster is used for training. The FedNIP part variation selects the best 10% of clients in the cluster and random 10% of clients in the cluster. Then the best weights of a client in a cluster is used, and the ranking is updated if appropriate.

Below in figure 7 the convergence rates of FedAvg (red), FedNIP full (blue), FedNIP part (green) and FedProx (purple) have been plotted for 50 clients with an alpha level of 0.6. Both the performance of FedAvg and FedProx is increasingly rapidly until round 40, after which the accuracy starts increasing more slowly. Around round 200 rounds, convergence seems to be reached at an accuracy of 0.63 for both FedAvg and FedProx. FedNIP full and FedNIP part seems to reach convergence after round 300. Where FedNIP full reaches a maximum accuracy of 0.58 and FedNIP part reaches a maximum accuracy of 0.54. Hereby, FedNIP full is 2 times faster than FedAvg and 2.5 times faster than FedProx. FedNip part is 3.45 times faster than FedAvg and 4.35 times faster than FedProx. Both variations of FedNIP have more fluctuation compared to FedAvg and Fedprox, which is to be expected as different set of weights of the different clusters get passed to the global model every round.

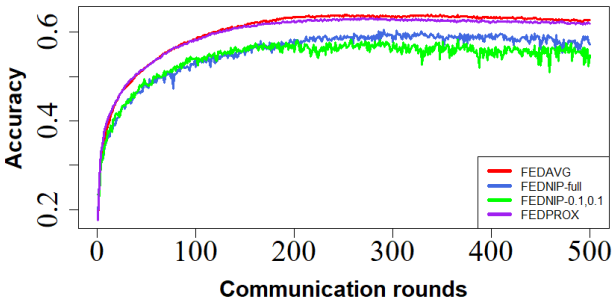

Figure 7: 50 clients with α of 0.6

In figure 8 the convergence rates of FedAvg (red), FedNIP full (blue), FedNIP part (green) and FedProx (purple) have been plotted for 50 clients with an alpha level of 0.3. One can observe that FedAvg and Fedprox are again similair in performance, where FedAvg looks more stable. The max accuracy of FedAvg is 0.6359. In this case, both FedNIP full and FedNIP part perform better compared to the alpha level of 0.6. FedNIP full has a max accuracy of 0.5995 and FedNIP part has a max accuracy of 0.5831. Hereby, FedNIP full is around 2 times faster than FedProx and FedAvg. Whereas, FedNIP part is 1.4 times faster than the full FedNIP.

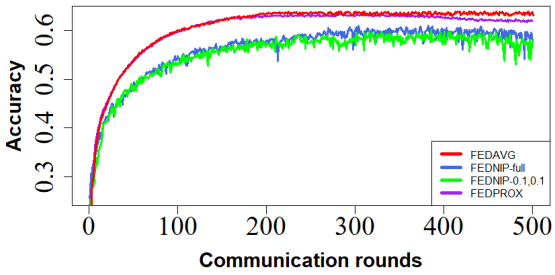

Figure 8: 50 clients with α of 0.3

In figure 9 the convergence rates of FedAvg (red), FedNIP full (blue), FedNIP part (green) and FedProx (purple) have been plotted for 50 clients with an alpha level of 0.05. This is the setting with the highest degree of statistical heterogeneity used in this research. Initially, Fedprox appears to be outperforming the other algorithms. However, both FedProx and FedAvg are decreasing after some time before stabilizing at around 500 communication round, the same level of FedNIP full and FedNIP part. The accuracy is in this case around 0.58. FedAvg and FedProx reach a higher maximum accuracy compared to FedNIP with respectively 0.605 and 0.62. However, they all convergence at the same level. Hereby, FedNIP full is 5.5 times as fast as FedProx and 4.5 times as fast as FedAvg. FedNIP part is more than 8 times faster than FedProx and 7 times faster than FedAvg. FedNIP part is 1.5 times faster than FedNIP full.

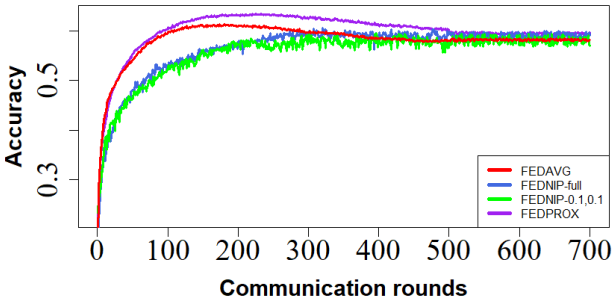

Figure 9: 50 clients with α of 0.05

In figure 10 the convergence rates of FedAvg (red), FedNIP full (blue), FedNIP part (green) and FedProx (purple) have been plotted for 100 clients with an alpha level of 0.6. FedProx and FedAVG have an almost identical performance. At around round 500 convergence seems to be reached with an accuracy of 0.6022. FedNIP full convergences around round 440 with an accuracy of 0.57. For FedNIP part, the convergence point is hard to determine as it fluctuates more heavily. Here, FedNIP full is almost 5 times as fast as FedProx and FedAvg.

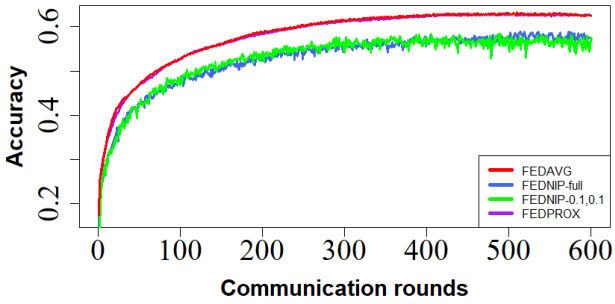

Figure 10: 100 clients with α of 0.6

In figure 11 the convergence rates of FedAvg (red), FedNIP full (blue), FedNIP part (green) and FedProx (purple) have been plotted for 100 clients with an alpha level of 0.3. FedProx is performing slightly better than FedAvg. Both FedAvg and FedProx, reach convergence at around round 500. FedNIP part is performing well in this case and keeps up with the FedNIP full algorithm. FedProx is reaching an accuracy of 0.62 when it reaches convergence. Both FedNIP full and FedNIP part reach an accuracy around 0.5776 when convergence is reached. Hereby, FedNIP full is 3 times faster compared to FedProx and FedAvg. FedNIP part is 4.5 times as fast as FedProx and FedAvg.

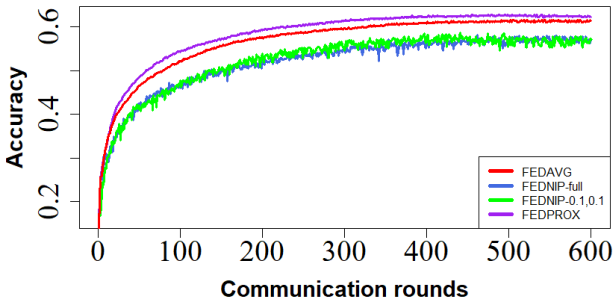

Figure 11: 100 clients with α of 0.3

In figure 12 the convergence rates of FedAvg (red), FedNIP full (blue), FedNIP part (green) and FedProx (purple) have been plotted for 100 clients with an alpha level of 0.3. FedProx is performing slightly better than FedAvg. Both FedAvg and FedProx, reach convergence at around round 500. FedNIP part is performing well in this case and keeps up with the FedNIP full algorithm. FedProx is reaching an accuracy of 0.62 when it reaches convergence. Both FedNIP full and FedNIP part reach an accuracy around 0.5776 when convergence is reached. Hereby, FedNIP full is 3 times faster compared to FedProx and FedAvg. FedNIP part is 4.5 times as fast as FedProx and FedAvg.

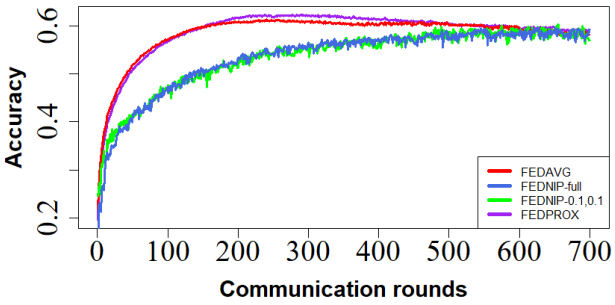

Figure 12: 100 clients with α of 0.05

In figure 13 the convergence rates of FedAvg (red), FedNIP full (blue), FedNIP part (green) and FedProx (purple) have been plotted for 100 clients with an alpha level of 0.05. FedAvg and FedProx reaches convergences around 600 rounds. FedNIP full and FedNIP part follow a similar pattern, where convergence is reached after 550 rounds. After 600 rounds they follow a similar trajectory where an accuracy is reached around 0.59. Hereby, FedNIP full is 3.7 times faster compared to FedProx. FedNIP part is even 7.4 times faster than FedProx.

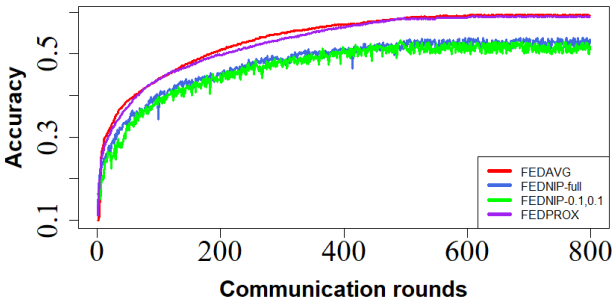

Figure 13: 250 clients with α of 0.6

In figure 14 the convergence rates of FedAvg (red), FedNIP full (blue), FedNIP part (green) and FedProx (purple) have been plotted for 250 clients with an alpha level of 0.6. FedAvg and FedProx reaches convergences just before 600 rounds. FedNIP full and FedNIP part follow a similar pattern, where convergence is also reached just before 600 rounds. Hereby, FedNIP full is 3.7 times faster compared to FedProx. FedNIP part is 7.4 times faster than FedProx.

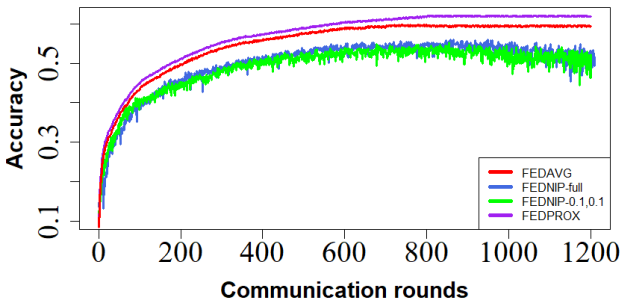

Figure 14: 250 clients with α of 0.3

In figure 14 the convergence rates of FedAvg (red), FedNIP full (blue), FedNIP part (green) and FedProx (purple) have been plotted for 250 clients with an alpha level of 0.3. FedAvg reaches convergence around round 630 with an accuracy of 0.57. FedProx reached convergence at round 800 at an accuracy of 0.605. FedNIP part follows a similar trajectory as FedNIP full, where convergence is reached around 600 at an accuracy of 0.53. Hereby, FedNIP full is 4 times faster compared to FedProx. FedNIP part is 7 times faster than FedProx.

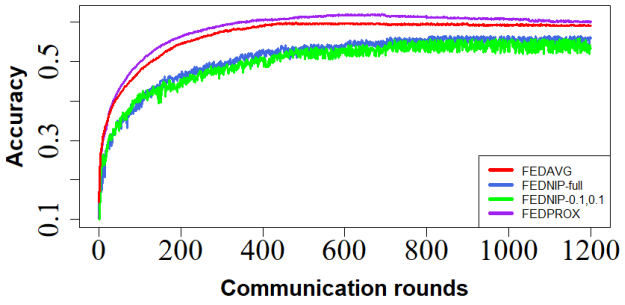

Figure 15: 250 clients with α of 0.11

In figure 15 the convergence rates of FedAvg (red), FedNIP full (blue), FedNIP part (green) and FedProx (purple) have been plotted for 100 clients with an alpha level of 0.11. Unfortunately, the petitioner function takes too much time to split on an alpha level of 0.05 with 250 clients. Therefore, the alpha level is set to 0.11. FedProx is performing the best and reached convergence at 1000 rounds at an accuracy of 0.58. FedAvg reaches convergence after 900 rounds at an accuracy of 0.56. FedNIP part is performing slightly worse compared to FedNIP full. Slightly before 800 rounds, convergence is reached of FedNIP full and FedNIP part. Hereby, FedNIP full is 5 times faster compared to FedProx and 6 times faster than FedAvg. FedNIP part is 8 times faster compared to FedProx and 10 times faster than FedAvg.

7 DISCUSSION

FedNIP has suboptimal performance for various values of α (0.6, 0.3, and 0.11) when compared to FedAvg and FedProx. This is likely due to the low degree of non-identically independent distributed data in which the data is being reasonably balanced. This means that there is increased bias in the training process compared to FedProx and FedAvg, where all clients participate, and weighted averaging is employed. However, in scenarios with a high degree of statistical heterogeneity and α set to 0.05, both variations of FedNIP (FedNIP full and FedNIP part) seem to achieve performance levels similar to FedProx and FedAvg.

Notably, FedNIP full is six times faster with 50 clients and three times faster with 100 clients compared to FedAvg, while FedNIP part is at least five times as fast. The increase in accuracy of FedNIP is initially slower, which is logical as FedNIP selects a cluster in a communication round and uses this for training, as opposed to including all the clients in a communication round. The variation of FedNIP (part), especially in highly statistically heterogeneous data scenarios where only 20% of clients in a cluster are selected each round, generally has similar or slightly worse performance compared to the full variant of FedNIP. This trend persists as the number of clients increases from 50 to 100 to 250, suggesting that the strategy of selecting a portion of top-performing and random clients represents a fair cluster representation.

The variance in FedNIP part is higher compared to FedNIP full, as expected, given that FedNIP part selects only a subset of clients for training on the proxy model, leading to higher variability in performance. In cases of high statistical heterogeneity, both FedAvg and FedProx experience a prolonged decrease in performance after certain rounds, likely due to model divergence caused by non-identically independent distributed (Non-IID) data, which is inevitable in highly Non-IID scenarios.
Unfortunately, the runtime of splitting on 250 clients according to the Dirichlet distribution with an α of 0.05 was too long, leaving uncertainties about FedNIP's performance under this condition. However, one can notice that FedNIP with an of α 0.11 reaches a higher performance compared to the α set to 0.3. Possibly indicating that with lower levels of α FedNIP also performs better also for 250 clients.

An important limitation of the Elbow method used in this study is that determining the optimal number of clusters for grouping based on similar data distribution becomes increasingly challenging with large datasets. Calculating the EMD becomes computationally expensive as it calculates all the distances between all pair of clients. Hereby, it also makes use of K-means clustering to determine the optimal value for K. For complex datasets it might not be so obvious with overlapping or irregularly shaped clusters. Leading to suboptimal number of clusters. This could possibly be mitigated by also integrating the Silhouette Score, which is an additional criterion for determining the optimal number of clusters.
Moreover, the number of tested parameters is limited. More parameters such as the number of hidden layers, local epochs, learning rate, mu of FedProx, k (percentage of top performing clients) and r (percentage of random selected clients) can be adjusted to further improve the performance.
There are also practical limitations of the current implementation of FedNIP. In practise, new clients can occur or the underlying data distribution of a client could change. In order to deal with this, a strategy could be implemented to periodically reevaluate the clusters to account for potential shifts in data distributions. This can be triggered based on certain events, such as the addition of new clients or a significant change in an existing client's data.
While the FedNIP method addresses privacy concerns by not sending raw data to the server and instead shares the data distributions, there is still a risk involved with the transfer of this information. If the server is comprised, a malicious person might identify individual clients based on the data distribution. This could be mitigated by adding noise to the data distribution when it gets sent to the server. Making sure that the statistical properties of the data distributions remain intact, while removing specific details that could potentially lead to the identification of individual clients.

Future research should focus on integrating cluster performance as a sampling criterion for each round, instead of the current client proportion-based sampling strategy. In order to prevent overfitting, one could use a similar approach as the core strategy of FedNIP: selecting top performing clusters and random performing clusters and train them on the proxy model and use the best results of a cluster.
Furthermore, FedNIP is now implemented as overall strategy. Further research could focus more on the client level. For example, creating heatmaps of the label distribution and occurrences after X number of communication rounds to study the effect on clients that do not have a lot of labels of a certain class.
One could also integrate other algorithms within FedNIP. For example, RepFL, could be used as selection criteria for the ranking of a client within a cluster based on the reputation that a client has built.
Lastly, it would also be interesting to assess FedNIP's performance in real-time environments, given its capability to dynamically rank clients based on performance.

8 CONCLUSION

In this study, the effectiveness of FedNIP, a new introduced algorithm, is examined in comparison to two widely used Federated Learning algorithms: FedAvg, extensively employed in various contexts, and FedProx, the established benchmark algorithm designed for Non-IDD data.

Hereby the following main question is investigated: how does the performance of FedNIP, in terms of global accuracy and speed, compare to that of baseline classifiers FedAvg and FedProx for clients with statistical heterogeneous data?

Two variations of FedNIP were used for experimentation. FedNIP full: meaning that all the clients in a cluster participate in training with the proxy model and FedNIP part where only the top 10% clients and 10% random clients get selected to train on the proxy model.

The performance of FedNIP is worse compared to FedAvg and FedProx in low to moderate degree of statistical heterogeneous data. This is likely due to the low degree of non-identically independent distributed data in which the data is being reasonably balanced. This means that there is increased bias in the training process compared to FedProx and FedAvg, where all clients participate, and weighted averaging is employed. Since, FedNIP uses only the weights of one client in a cluster in a given communication round. This observation holds true regardless of the scaling of the number of clients.

In high statistical environments with an α of 0.05 FedNIP outperforms FedAvg, the most established FL algorithm, and matches FedProx, the most established Non-IID FL algorithm. This observation also holds true regardless of the scaling of the number of clients.

Using the FedNIP strategy of only using a subset of the clients (using top 10% performing clients in a cluster and 10% of random clients in a cluster) has similar performance compared to using FedNIP where all the clients are utilized in a cluster.

This outcome means that examining a subset of clients within a cluster provides a reliable indication of the overall performance of the entire cluster. Which drastically reduces the number of needed clients to be used. FedNIP's runtime is four to eight times faster than FedAvg and FedProx, dependent on number of clients and level of statistical heterogeneity.

REFERENCES

[1] Zexin Sha Guoming Lu Aiguo Chen, Yang Fu. 2022. An EMD-Based Adaptive Client Selection Algorithm for Federated Learning in Heterogeneous Data Scenarios. *Proceedings of the 29th Conference on Neural Information Processing Systems (NeurIPS)* (6 2022), 1–21. https://doi.org/10.3389/fpls.2022.908814

[2] Daniel Beutel. 2020. Flower FL. https://flower.dev/docs/tutorial/Flower-0-What-is-FL.html. [Online; accessed 4-April-2023].

[3] Ed Burns. 2018. Deep Learning. https://www.techtarget.com/searchenterpriseai/definition/deep-learning-deep-neural-network. [Online; accessed 4-April-2023].

[4] Yi Zhou Bhavya Kailkhura Cheng Chen, Ziyi Chen. 2020. FedCluster: Boosting the Convergence of Federated Learning via Cluster-Cycling. *Proceedings of the 29th Conference on Neural Information Processing Systems (NeurIPS)* (9 2020), 1–10. https://doi.org/10.48550/arXiv.2009.10748

[5] Francesco Malandrino; Carla Fabiana Chiasserini. 2021. FEDBS: Learning on Non-IID Data in Federated Learning using Batch Normalization. *Journals Magazines >IEEE Communications Magazine* (7 2021), 1–38. https://doi.org/10.1109/ICTAI52525.2021.00138

[6] Dihuni. 2020. 2.5 Quintillion Bytes of Data Generated Everyday Top Data Science Trends 2020. https://www.dihuni.com/2020/04/10/every-day-big-data-statistics-2-5-quintillion-bytes-of-data-created-daily/, Last accessed on 2023-02-12.

[7] Hyesung Kim Jihong Park Mehdi Bennis Seong-Lyun Kim Eunjeong Jeong, Seungeun Oh. 2018. Communication-Efficient On-Device Machine Learning: Federated Distillation and Augmentation under Non-IID Private Data. *IEEE International Conference on Communications* (10 2018), 1–41. https://doi.org/10.48550/arXiv.1811.11479

[8] Daniel Ramage Blaise Agüera y Arcas H. Brendan McMahan, Eider Moore and John Langford. 2017. Communication-Efficient Learning of Deep Networks from Decentralized Data. *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics (AISTATS)* 54 (4 2017), 1273–1282. https://doi.org/10.48550/arXiv.1602.05629

[9] Zijing Duan Wei Guo Jianhang Xiao, Chunhui Du. 2021. A Novel Server-side Aggregation Strategy for Federated Learning in Non-IID situations. *Journals Magazines >IEEE Communications Magazine* (4 2021), 1–38. https://doi.org/10.1109/ISPDC52870.2021.9521631

[10] Hao Liang Gauri Joshi H. Vincent Poor Jianyu Wang, Qinghua Liu. 2021. Tackling the Objective Inconsistency Problem in Heterogeneous Federated Optimization. *Journals Magazines >IEEE Communications Magazine* (4 2021), 1–38. https://doi.org/10.48550/arXiv.2007.07481

[11] Wei Chen Tie-Yan Liu Jie Liu, Bo Li. 2020. FedDistill: Making Bayesian Model Ensemble Applicable to Federated Learning. *Proceedings of the 29th Conference on Neural Information Processing Systems (NeurIPS)* (10 2020), 1–21. https://doi.org/10.48550/arXiv.2009.01974

[12] Jessica Zhao Kavya Kopparapu, Eric Lin. 2021. FedCD: Improving Performance in non-IID Federated Learning. *Journals Magazines >IEEE Communications Magazine* (7 2021), 1–38. https://doi.org/10.48550/arXiv.2006.09637

[13] Li Li Yingwen Chen Ming Xu Cheng-Zhong Xu Liang Gao, Huazhu Fu. 2022. FedDC: Federated Learning with Non-IID Data via Local Drift Decoupling and Correction. *IEEE International Conference on Communications* (10 2022), 1–41. https://doi.org/10.48550/arXiv.2203.11751

[14] Sue Liu. 2022. Dirchelet distribution. https://builtin.com/data-science/dirichlet-distribution. [Online; accessed 4-April-2023].

[15] Xiao Wang Qi Zhu Lixu Wang, Shichao Xu. 2020. Addressing Class Imbalance in Federated Learning. *Journals Magazines >IEEE Communications Magazine* (4 2020), 1–38.

[16] Sambit Mahapatra. 2018. Why Deep Learning over Traditional Machine Learning? https://towardsdatascience.com/why-deep-learning-is-needed-over-traditional-machine-learning-1b6a99177063. [Online; accessed 4-April-2023].

[17] ZezhongMa MengyingZha MeiCao, YujieZhang. 2022. Class-aware client selection for effective aggregation in federated learning. *Proceedings of the 29th Conference on Neural Information Processing Systems (NeurIPS)* (9 2022), 1–7. https://doi.org/10.1016/j.hcc.2022.100068

[18] Roger Roberts Michael Chui and Lareina Yee. 2022. McKinsey Technology Trends Outlook 2022.

[19] Xianzhang Chen Yujuan Tan Jinting Ren Lei Qiao Liang Liang Moming Duan, Duo Liu. 2019. Astraea: Self-balancing Federated Learning for Improving Classification Accuracy of Mobile Deep Learning Applications. *IEEE International Conference on Communications* (10 2019), 1–41. https://doi.org/10.48550/arXiv.1907.01132

[20] Masahiro Morikura Koji Yamamoto Ryo Yonetani Naoya Yoshida, Takayuki Nishio. 2020. Hybrid-FL for Wireless Networks: Cooperative Learning Mechanism Using Non-IID Data. *Journals Magazines >IEEE Communications Magazine* (4 2020), 1–38. https://doi.org/10.48550/arXiv.1905.07210

[21] Gavril Ognjanovski. 2019. Everything you need to know about Neural Networks and Backpropagation. https://towardsdatascience.com/everything-you-need-to-know-about-neural-networks-and-backpropagation-machine-learning-made-easy-e5285bc2be3a. [Online; accessed 4-April-2023].

[22] Mehryar Mohri Sashank J. Reddi Sebastian U. Stich Ananda Theertha Suresh Sai Praneeth Karimireddy, Satyen Kale. 2019. SCAFFOLD: Stochastic Controlled Averaging for Federated Learning. *IEEE International Conference on Communications* (10 2019), 1–41. https://doi.org/10.48550/arXiv.1910.06378

[23] Manzil Zaheer Sashank J. Reddi, Zachary Charles. 2021. ADAPTIVE FEDERATED OPTIMIZATION. *Journals Magazines >IEEE Communications Magazine* (4 2021), 1–38.

[24] SG Analytics. 2020. 2.5 Quintillion Bytes of Data Generated Everyday Top Data Science Trends 2020. https://us.sganalytics.com/blog/2-5-quintillion-bytes-of-data-generated-everyday-top-data-science-trends-2020//, Last accessed on 2023-02-12.

[25] Danny Tsang Tailin ZHOU, Jun Zhang. 2023. FedFA: Federated Learning with Feature Alignment for Heterogeneous Data. *Journals Magazines >IEEE Communications Magazine* (4 2023), 11–16. https://doi.org/10.48550/arXiv.2007.07481

[26] Ryo Yonetani Takayuki Nishio. 2019. Client Selection for Federated Learning with Heterogeneous Resources in Mobile Edge. *Proceedings of the 29th Conference on Neural Information Processing Systems (NeurIPS)* (10 2019), 1–7. https://doi.org/10.48550/arXiv.2003.00295

[27] Manzil Zaheer Maziar Sanjabi Ameet Talwalkar Virginia Smith Tian Li, Anit Kumar Sahu. 2018. FEDERATED OPTIMIZATION IN HETEROGENEOUS NETWORKS. *Proceedings of the 29th Conference on Neural Information Processing Systems (NeurIPS)* (12 2018), 1–21. https://doi.org/10.48550/arXiv.1812.06127

[28] Standford University. 2022. Earth Movers Distance. http://infolab.stanford.edu/pub/cstr/reports/cs/tr/99/1620/CS-TR-99-1620.ch4.pdf. [Online; accessed 4-April-2023].

[29] Toronto University. 2014. CIFAR-10 Dataset. https://www.cs.toronto.edu/~kriz/cifar.html. [Online; accessed 4-April-2023].

[30] Péter Kiss Vukasin Felbab and Tomáš Horváth. 2020. Optimazation in Federated Learning. https://ceur-ws.org/Vol-2473/paper13.pdf. [Online; accessed 4-April-2023].

[31] Yuwei Wang and Burak Kantarci. 2021. Reputation-enabled Federated Learning Model Aggregation in Mobile Platforms. *IEEE International Conference on Communications* (6 2021), 1–11. https://doi.org/10.1109/ICC42927.2021.9500928

[32] Kesper Welbers. 2021. alpha parameter in Dirchelet distribution. https://i.amcat.nl/lda/understanding_alpha.html. [Online; accessed 4-April-2023].

[33] Pan Zhou Wenyu Zhang, Xiumin Wang. 2021. Client Selection for Federated Learning With Non-IID Data in Mobile Edge Computing. *Journals Magazines >IEEE Communications Magazine* (6 2021), 24462 – 24474. https://doi.org/10.1109/ACCESS.2021.3056919

[34] Zhihao Lin Shanxuan Chen Yangjie Qin Xiaodong Ma, Jia Zhu. 2020. A state-of-the-art survey on solving non-IID data in Federated Learning. *Journals Magazines >IEEE Communications Magazine* (4 2020), 1–38. https://doi.org/10.1016/j.future.2022.05.003

[35] Xian Shuai; Yulin Shen; Siyang Jiang; Zhihe Zhao; Zhenyu Yan; Guoliang Xing. 2022. BalanceFL: Addressing Class Imbalance in Long-Tail Federated Learning. *International Conference on Information Processing in Sensor Networks* (9 2022), 1–14. https://doi.org/10.48550/arXiv.2009.10748

[36] Wenqi Li Yingda Xia, Dong Yang. 2021. Auto-FedAvg: Learnable Federated Averaging for Multi-Institutional Medical Image Segmentation. *International Conference on Information Processing in Sensor Networks* (4 2021), 1–11. https://doi.org/10.48550/arXiv.2009.10748

[37] Syed Zawad Zheng Chai, Ahsan Ali. 2020. TiFL: A Tier-based Federated Learning System. *Journals Magazines >IEEE Communications Magazine* (6 2020), 125–136.

[38] Lei Tan; Xiaoxi Zhang; Yipeng Zhou. [n. d.]. ([n. d.]). https://doi.org/10

[39] Weiming Zhuang and Lingjuan Lyu. 2023. Recent Breakthroughs Tackle Challenges in Federated Learning. *Privacy-Preserving Machine Learning Blog Series* (2023).