Peer to Peer Federated Learning: Towards Decentralized Machine Learning on Edge Devices

Aristeidis Karras*, Christos Karras*, Konstantinos C. Giotopoulos†, Dimitrios Tsolis*, Konstantinos Oikonomou‡, Spyros Sioutas*
*Decentralized Systems Computing Group, Computer Engineering and Informatics Department, University of Patras, Patras, Greece
{akarras, c.karras, sioutas}@ceid.upatras.gr, dtsolis@upatras.gr
†Department of Management Science and Technology, University of Patras, Patras, Greece
kgiotop@upatras.gr
‡Department of Informatics, Ionian University, Corfu, Greece
okon@ionio.gr

2022 7th South-East Europe Design Automation, Computer Engineering, Computer Networks and Social Media Conference (SEEDA-CECNSM) | 979-8-3503-9858-8/22/$31.00 ©2022 IEEE | DOI: 10.1109/SEEDA-CECNSM57760.2022.9932980

***Abstract*—Federated Learning (FL) is an emerging technique that assures user privacy and data integrity in distributed machine learning environments. To perform so, chunks of data are trained across edge devices and a high performance cluster server maintains a local copy without exchanging it with other parties. In this work, we investigate a FL scenario in a real-world case study using 5, 10 and 20 Raspberry Pi devices acting as clients. Under this setup, we employ the widely known FedAvg algorithm which trains each client for several local epochs and then the weight of each model is aggregated. Moreover we perform experiments across imbalanced and noisy data so as to explore scalability and robustness based on real-world datasets were noise is present and we also propose two innovative algorithms where the FL scenario is considered as a peer-to-peer formulation. Ultimately, to ensure that each device is not over-sampled a client-balancing Dirichlet sampling algorithm with probabilistic guarantees is proposed.**

***Index Terms*—Decentralized Machine Learning, Federated Learning, P2P FL, Client Balancing Dirichlet Sampling**

I. Introduction

With the rapid emerge of Artificial Intelligence there is vast amount of applications that are developed within the scope of autonomous systems and to aid in peoples lives. However, this rapid trend has created serious concerns regarding personal privacy [1]. Therefore, large tech companies such as Google and Apple have turned to Federated Learning (FL) techniques that enhance privacy among widely-used applications including Siri, Google Chrome and Gboard [2]. In particular, the term FL employs a distributed machine learning technique that allows training to occur across several machines on a vast repository of decentralised data existing on devices like mobile phones [3]. Then, a central server aggregates these local updates [4]. This process solves the core issues of exposing data privacy, ownership, and location. The aggregation process takes place by iteratively averaging weights from models, via training each client for several local epochs. This process is known as the formative *FedAvg* [4] algorithm. For faster and more stable convergence, additional aggregation methods have been developed, including the ensemble distillation algorithm namely *FedDF* [5].

Within a FL scheme, there is always the dilemma whether to focus on asynchronous or synchronous training algorithms. Comprehensive work on deep learning has used asynchronous training while other works as in [6], have proven that there is a consistent trend towards synchronous large batch training, even occurring within data-centers. The Federated Averaging (*FedAVG*) algorithm operates in a similar way. Furthermore, several approaches for enhancing privacy guarantees including differential privacy [7] and secure aggregation [8], essentially require some notion of synchronization on a fixed set of devices, so that the server-side of the learning algorithm only consumes a simple aggregate of updates from many users.

In related works, many benchmarks of FL performance over algorithmic choices exist but are often performed by simulating the federation on central compute clusters [5]. In this work, we aim to capture the unique hardware setup of FL use-cases such as smartphones, where a number of computationally weak edge devices where the data is stored. This is accomplished by conducting local training of a convolutional neural network (CNN) on 10 Raspberry Pi devices across which CIFAR-10 is distributed, and then centrally aggregated on these locally trained models using *FedAvg* and *FedDF* methods.

This work intends to examine the effect of FedAvg hyper-parameters, such as the number of clients per communication round and local epochs, on convergence time, analyse aggregation resilience against unbalanced and noisy data, and identify performance trade-offs in the physical hardware scenario. Moreover, a fine-tuning to traditional FL takes place using two methods: a) using a p2p network where the weights client-processed and b) using a method for p2p federated learning.

The remainder of the paper is organized as follows. In Section II related work in the field of Federated Learning is surveyed and covered. Section III describes the methodology in both theory and application level while Section IV highlights the experimental results and their findings. A discussion takes place in V and finally, the conclusions and future directions of this work are presented in Section VI.

979-8-3503-9858-8/22/$31.00 ©2022 IEEE

II. Related Work

A. Development of Federated Learning

Federated learning (FL) although appears to be a novel architecture for protecting privacy among users, is sometimes not understood by the general public. Therefore, the examples presented in this section highlight the inner workings of FL for better understanding. Suppose that a certain number of distinct businesses are willing to cooperate on a machine learning model training process [3]. Based on the GDPR criteria, the data utilized by both parties cannot be used in the absence of the consent of their respective users. In contrast, a business may develop a model for machine learning using their own data stored locally. Assuming that each of the involved parties constructs a task model, it may be challenging to construct and train an optimal machine learning model because of limited and sometimes noisy data. The aim of FL is to address the aforementioned issues by guaranteeing that the local data of each organisation remains internal. Utilizing an encryption technology, parameters are transferred among clients and the central server so as to develop a global model in order to not violate the laws regulating the protection of privacy.

B. Horizontal Federated Learning

Horizontal federated learning (HFL) is a sub-sector of FL that can be utilized in case the characteristics of a user across two given datasets significantly overlap, but the users themselves do not. HFL entails slicing datasets horizontally (across the user factor) and removing for training the portion of data whose user attributes are same but whose users are not identical. In particular, distinct rows containing the data have identical characteristics and HFL may thereby extend the sample size of users.

For example, there might be two suppliers offering the same service in separate locations whose user groups are mostly comprised of individuals from their respective regions, with minimal overlap. Nevertheless, their companies are really comparable, thus the user characteristics of the records are as well. In this regard, we may train a model using horizontal FL, which besides that it increases the total amount of training samples, it also improves the overall accuracy achieved by the model. In the case of HFL, it is feasible that all participants compute and submit local elements for the central server to combine them and construct a global model. Moreover, the procedure of processing and transferring elements may expose private information of the users. Widely used solutions for the aforementioned issue include homomorphic encryption schemes [9], differential privacy methods [7], and secure aggregation [10] frameworks, which may guarantee the safety of gradient switching in HFL.

A FL modelling scheme built for Android phones was proposed in 2016 by Google [8] in which, a user occupying an Android phone is able to update the parameters of the model in constant time locally, and then to upload the parameters onto the cloud, enabling all the owners of the data under the corresponding feature dimension to develop a FL model. The aforementioned framework comprises of a fundamental implementation of HFL, including differential privacy [7] and secure aggregation techniques. An HFL system namely BlockFL is introduced in [11], whereabouts each device leverages the widely used blockchain network to update its local learning model. In [12], the authors suggest a FL technique named MOCHA to overcome multitasking security issues, which enables several sites to collaborate and perform tasks while maintaining privacy and security. Another method, namely multi-task FL also reduces the communication cost and increases the fault tolerance of the original distributed multi-task learning method. In [4], FL separates the data, enabling the corresponding client to prevent uploading of sensitive information to a central-server. Then, every client constructs a local model, submits it to the server, and stores a copy of a global gradient-based model.

C. Vertical Federated Learning

In the use-case that characteristics of users across two given datasets overlap slightly, but the users overlap a lot, a common solution to overcome this matter is vertical federated learning (VFL). In particular, it involves dividing the given datasets vertically in a user/feature dimension, removing for training purposes the portion of data where users are the same but user characteristics differ. In particular, the same user is represented in many columns. VFL may thereby expand the training data based on the feature dimension.

For instance, there might be two distinct companies, a bank in one location and a brand for e-commerce in the same exact location. The user groups of such companies are likely to contain the majority of local population, resulting in a larger overlap of users. Due to the fact that banks record the income of users, their spending behaviour and credit rating, while e-commerce stores the users browsing and purchase histories, their user characteristics are largely different. VFL is the aggregation of various diverse features in a secure state to improve the performance of the model. Numerous machine learning models, including the logistic regression model, the tree structure model, and the neural network model, have been shown to be built on this federated system.

In order to perform vertical data partitioning, there is a vast amount of machine learning techniques including classification [13], statistical analysis [14], gradient descent [15], linear regression for privacy [16], and privacy-based data mining techniques [17]. In certain instances of VFL, there are also vertically partitioned data. In [18], a VFL system named SecureBoost is suggested, whereabouts all members contribute user characteristics to train jointly in order to increase the precision of decision-making. This training scheme is lossless. In [19], a suggested privacy-protected logical regression model based on VFL is introduced. The method utilises parallelizing objects for analysis and performs logistic regression in a distributed manner for supplementary homomorphic encryption [20], which aids in privacy protection and also improves the accuracy of the classifier.

III. Methodology

A. Federated Learning Methods

In this section federated learning is explored and its implementation. In particular, FL is implemented by setting up K clients each with a disjoint training dataset partition of size n_k across a larger dataset. A global model namely $\mathcal{M}_G$, was deployed and stored across a central server, and for L rounds, defined as *communication rounds*, $S \leq K$ clients were sampled using a Dirichlet sampling algorithm, each of which received a local copy of $\mathcal{M}_G$. Each client performed E local epochs of learning before returning the updated local model $\mathcal{M}_k$ to the server. In the initial FedAvg algorithm, the server aggregated the S returned models by averaging over all model weights, yielding a new global model to be sent out for the next communication round, minimizing an implicit objective function f of model weights w when using a loss function ℓ:

$$f(\mathbf{w}) = \sum_{k=1}^{K} \frac{n_k}{n_{\text{total}}} F_k(\mathbf{w}), F_k(\mathbf{w}) = \frac{1}{n_k} \sum_{i=1}^{n_k} \ell(x_i, y_i; \mathbf{w}), \quad (1)$$

where $F_k(\mathbf{w})$ is the objective function of the k-th client [4].

A different aggregation method, namely the FedDF algorithm was implemented as well, where the weight averaging method of FedAvg is replaced by running ensemble distillation for model fusion using an unlabeled dataset similar to the training datasets. Using similar hyperparameters as in [5], $\mathcal{M}_G^{(l+1)}$ was produced by distilling the S local models $\mathcal{M}_{k_i}^{(l)}$ for 10^4 batch updates against a Kulback-Leibler (KL) divergence criterion with a batch size of 128 and a learning rate of 10^{-3} used for Adam optimization with cosine annealing. Early stopping was implemented by calculating the KL divergence against an unlabeled validation set every 10^3 updates and terminating if evaluation loss did not fall. The KL divergence criterion is further described in Section III-G.

B. Embedded Devices

The engineering part of this work shown in Figure 1, was divided into two parts: a central high-performance cluster (HPC) and 10 Raspberry Pi model 4 devices, each with 4 GB memory and a quad-core 1.5 GHz CPU. In particular, the Pi devices were connected on a different network from the HPC to simulate a more realistic communication scenario. The HPC server was responsible for aggregating the local models trained by the devices and evaluating the resulting global model $\mathcal{M}_G$.

Every Raspberry Pi hosted a Flask server that exchanged models, performed local training, and reported running memory usage; a crucial factor when operating on such resource-constrained hardware. Moreover, the server provided a route for transmitting requests to enable basic over-the-air updates.

The Raspberry Pi setup was able to run experiments where $K > 10$, that is, maintaining more than 10 clients and thus dataset partitions with the constraint of no more than 10 clients being sampled each round: $S \leq 10$. This was implemented by storing all K client datasets on every device, and in each communication round l, assigning each of the S sampled client datasets to a Raspberry Pi.

The Pi devices were attached to a switch that was cable-connected to the router. When the switch was turned off, the Pis were configured to connect through Wi-Fi, enabling measurements of the effect of communication overhead under two conditions: reasonably fast Ethernet and relatively sluggish Wi-Fi. In particular, the network used had a bandwidth of 100/100 Mbit/s, all of which was utilised on Ethernet, but only about 10% (10/10 Mbit/s) on Wi-Fi. Each client in Figure 1 corresponds to a dataset partition $k_1 \dots k_S$ where Raspberry Pi 1 trains a model as client k_i, and Raspberry Pi 10 as k_j.

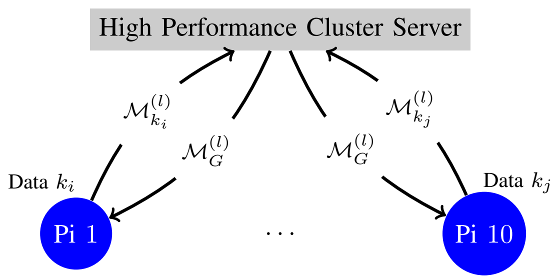

High Performance Cluster Server
$\mathcal{M}_{k_i}^{(l)}$
$\mathcal{M}_G^{(l)}$
$\mathcal{M}_G^{(l)}$
$\mathcal{M}_{k_j}^{(l)}$
Data k_i
Data k_j
Pi 1
...
Pi 10

Fig. 1: The federated setup performing updates at communication round l, at which $S \leq 10$ clients are sampled.

C. Problem formulation and dataset

To deploy the federated learning scenario described previously, we select the CIFAR-10 computer vision (CV) dataset that contains 32×32 images for image classification into object classes such as birds, cats, and aeroplanes. Due to device memory limits and for faster training times all images used, were grey-scaled. The training dataset contains 5K images for each of the 10 label. For the model $\mathcal{M}$, we chose a network with two convolutional layers followed by two linear layers, which is further detailed in Table I.

The model architecture is described below, listing the sequential operations and hyperparameters.

TABLE I: Model Architecture and layers.

Layer type	Hyperparameters
2D Convolution	1 in-channel, 16 out-channels, 3×3 kernel, stride of 1
2D Convolution	16 in-channels, 32 out-channels, 3×3 kernel, stride of 1
2D MaxPooling	2×2 kernel, stride of 2, no padding, dilation of 1
Dropout	$p = 25\%$
Linear w. bias	$6,272$ features in, 64 features out
Dropout	$p = 50\%$
Linear w. bias	64 features in, 10 features out

The model was limited in size to accommodate the strict memory limits on the Raspberry Pi devices. Using the Adam optimizer with a learning rate η, which was decayed every local epoch: $\eta \leftarrow \gamma\eta, \gamma \leq 1$, optimization for the E local epochs on each device was performed.

D. Handling unbalanced and noisy data

1) Dirichlet sampling: The total training dataset was divided into K evenly sized partitions among all the clients. In practice, class balance can rarely be assumed in the FL setting [2]. In order to simulate varying levels of imbalance, the Dirichlet distribution, $\mathrm{Dir}(\boldsymbol{\alpha})$, was used. The length of the parameter vector $\boldsymbol{\alpha}$ corresponds to the number of labels, 10, and we let $\alpha_i = \alpha$. Every sample $\boldsymbol{\pi} \sim \mathrm{Dir}(\boldsymbol{\alpha})$ is a probability distribution over labels and α determines the uniformity of the corresponding distribution. For $\alpha \to 0$, one label dominates, where as for $\alpha \to \infty$, $\boldsymbol{\pi}$ will be increasingly uniform. For $\alpha = 1$, every possible $\boldsymbol{\pi}$ is equally likely to happen.

For every client, $\boldsymbol{\pi}$ was sampled making the label distribution Dirichlet for every client. To keep the client datasets disjoint while using a maximal part of the total dataset, we created a client-balancing Dirichlet sampling algorithm described in detail in Section III-E.

E. Client-Balancing Dirichlet Sampling Algorithm

Let D denote a dataset of size $|D|$ with l different labels, of which there are $|D|/l$ each. The goal of the algorithm is to divide the dataset among C clients, such that the label distribution, $\boldsymbol{\pi}_i$ of each client i, follows the same Dirichlet distribution, $\mathrm{Dir}(\boldsymbol{\alpha})$, where every $\alpha_j \in \boldsymbol{\alpha}, j = 1 \dots l$ is the same value. For simplicity, the Dirichlet distribution will therefore be parametrized only by α; $\mathrm{Dir}(\alpha)$.

The distributions are structured into a matrix $\mathbf{P}$ of size $C \times l$, where the i-th row is $\boldsymbol{\pi}_i$. The ij-th element, P_{ij}, is then the fraction of label j on device i. Furthermore, the sum of the j-th column is the relative usage of label j scaled by the number of clients, C. As such, for every label to be used equally much, every column should sum to C/l. If any columns sum is more, the corresponding labels are oversampled, and so all of $\boldsymbol{P}$ needs to be normalized to make the largest column-sum equal C/l, causing some of part of the data to be disregarded. Step 5 of the algorithm mapper swaps to u values while step 6 does all possible swaps. If there is no other possible swap, step 17 stops the process. Finally, step 21 normalizes $\mathbf{P}$ by the most sampled label. To determine how close $\mathbf{P}$ is to achieving the goal of making every column sum to the same a measure, u, is utilized. The lowest value of this measure should be achieved for a $\mathbf{P}$ where every column sums to C/l, while it should be progressively higher for poorer $\mathbf{P}$ values. We chose the L1 norm of difference in column sums and C/l:

$$u(\mathbf{P}) = \sum_{j=1}^{l} \left| \frac{C}{l} - \sum_{i=1}^{C} P_{ij} \right| \tag{2}$$

Similarly, standard deviation or the reciprocal of the entropy could be used. The final thing to keep in mind before the algorithm is introduced is that reordering $\boldsymbol{\pi}_i$ makes it equally likely to be sampled from $\mathrm{Dir}(\alpha)$.

The algorithm changes the ordering within each $\boldsymbol{\pi}_i$ iteratively, until $u(\mathbf{P})$ is no longer lowered. The full algorithm steps are shown in Algorithm 1. The final $\mathbf{P}$ produced by the algorithm is normalized such that all data points with the most sampled labels are used exactly once.

Mostly, labels were sampled more evenly for higher values of C and α. Running the algorithm for $C = 100$ and $\alpha = 0.01$ for 100 times indicated an average undersampling of less than 1% with the largest undersampling being 3.3%. Rarely was any one label undersampled by more than 5%, a sign of the strength of the algorithm, even for a low value of α.

Algorithm 1 Client-Balancing Dirichlet Sampling (CBDS)

```
1:  for $i$ from 1 to $C$ do
2:      $\mathbf{P}_i \leftarrow \mathrm{Dir}(\alpha)$
3:  end for
4:  loop
5:      $d \leftarrow$ Map()
6:      for $i$ from 1 to $C$ do
7:          for all $(j_1, j_2) \in (1 \dots l) \times (1 \dots l), j_1 \neq j_2$ do
8:              Swap $P_{ij_1}$ and $P_{ij_2}$
9:              $d[(i, j_1, j_2)] \leftarrow u(\mathbf{P})$
10:             Swap $P_{ij_1}$ and $P_{ij_2}$ back
11:         end for
12:     end for
13:     if $\min(d) < u(\mathbf{P})$ then
14:         $i, j_1, j_2 \leftarrow \mathrm{argmin}(d)$
15:         Swap $P_{ij_1}$ and $P_{ij_2}$
16:     else
17:         Break
18:     end if
19: end loop
20: $\boldsymbol{\pi} \leftarrow \sum_{i=1}^{C} \boldsymbol{\pi}_i$
21: $\mathbf{P} \leftarrow \mathbf{P} / \max(\boldsymbol{\pi})$
```

F. Extension to Federated Learning

1) Peer-to-peer: In this subsection one of the possible extensions to federated learning is presented, in particular, a peer-to-peer scenario. Generally in FL, users would send their weights and evaluations to a central agent C that would average their weights and return the averaged weights back to all the users/clients. The averaging is based on the averaging methodology being utilised by C. The averaged weights are then used by the users in their models for the next round of federated learning. But in a peer-to-peer context, the averaging process takes place on every user's device itself instead of C. Every user U_i in U sends its weights to every other user in U which means that every user U_i has access to the weights from every user in U. We call this set of weights received from other users $W_{\text{others.}}$ U_i then performs evaluations locally using its own local validation data on its own model M_i and on models initialised from every weight from the set $W_{\text{others.}}$. This way it can emulate models that the other users have learnt. Doing so, U_i can check how relevant everyone else's models are for its own data and take appropriate action depending on the averaging methodology being used. The evaluations obtained from this process are then used in the averaging process by U_i to calculate $W_{\text{avg.}}$. This W_{avg} is used in the next round of local learning by U_i. It is not shared with any other users. As W_{avg} is now specific to U_i, we call it $W_{i,avg}$.

Algorithms 2 and 3 present the proposed extension to the standard FL where devices act as users in a P2P scenario.

Algorithm 3 is a simulation of the peer-to-peer communication that would take place in a real world scenario where the

Algorithm 2 Peer-to-Peer Side Processing on Client

```
1:  def user(set_of_weights):
2:  if set_of_weights != None: then
3:      new_weights = local_average(set_of_weights)
4:      $M_i$.set_weights(new_weights)
5:  end if
6:  evaluation = device.evaluate_model ()
7:  device.add_pre_fit_evaluation(evaluation)
8:  for e in $local\_epochs$ : do
9:      $M_i$.train()
10: end for
11: evaluation = device.evaluate_model ()
12: device.add_post_fit_evaluation(evaluation)
```

Algorithm 3 Peer-to-Peer Federated Learning

```
1:  def federated_learning_p2p(model):
2:  send_to_devices(model)
3:  set_of_weights = None
4:  for $round$ in $rounds$: do
5:      for $device$ in $devices$: do
6:          device(set_of_weights)
7:          set_of_weights = []
8:      end for
9:      for $device$ in $devices$: do
10:         set_of_weights.append(device.weights)
11:     end for
12: end for
```

devices would broadcast their weights to one another. In the simulation, the algorithm acts as a coordinator for the process and takes care of sharing every device's weights with all devices by collecting every device's weights and sending the set of weights to everyone. The algorithm starts off by sending all the users in U the model M that they will be using in this process. Then, for every round of federated learning, we start off by training all the users which is illustrated in Algorithm 2. In this process, if the algorithm provides a set of weights to a user U_i, U_i will calculate the averaged weights based on the local averaging methodology and then initialise its model with the newly computed averaged weights. The averaging will be discussed more in depth in the following section. If no weights are provided to the user, then the weights of the user's model are not changed. The model is then evaluated on the U_i 's local validation data and the evaluation results are stored in the pre_fit evaluations. After doing so, the model is trained for a number of epochs and then once again evaluated on the U_i 's local validation data and the evaluation results are stored in the post_fit evaluations. After the training process, all the weights from the users are collected and sent to every user in the next round of federated learning. When the next round of federated process begins, the whole process is repeated. This is performed for a set number of federated learning rounds.

The implementation of this framework requires minimal changes with respect to the implementation of the central approach. The changes in implementation to cater to Algorithms 2 and 3 are explained here. After the train method is called on every U_i, we iterate over all devices again and create a dictionary mapping of user ID to their weights such that U_i 's ID i points to W_i. Device ID i can be accessed from U_i by using the get_id method. We call this dictionary id_to_weights. This dictionary is then passed into the train method of U_i in the next round of federated learning instead of the averaged weights which is a list of numpy arrays. In the train method of U_i, we now also have a check to see if the weights provided are in a dictionary data structure. If so, then the peer-to-peer learning process is being used and local averaging must be performed. So the dictionary is then passed into a method which is part of the class Average to compute the averaged weights $W_{i,avg}$ for U_i. After the local averaging method returns $W_{i,avg}$, the set_weights method is used to set the weights of M_i before local training commences.

G. Kulback-Leibler Divergence in FedDF Algorithm

The KL divergence measures the distance from one distribution, Q, to a reference distribution, P. Let P and Q be discrete distributions defined on the probability space $\mathcal{X}$. The KL divergence is then defined as

$$D_{\mathrm{KL}}(P\|Q) = \sum_{x\in\mathcal{X}} P(x)\log\left(\frac{P(x)}{Q(x)}\right) \tag{3}$$

A key property is $D_{\mathrm{KL}}(P\|Q) = 0 \Leftrightarrow P = Q$. However, the divergence is not a metric on the space of probability distributions, as in general $D_{\mathrm{KL}}(P\|Q) \neq D_{\mathrm{KL}}(Q\|P)$.

In the context of FedDF, the KL divergence uses the probability predictions of the student (or central) model as Q, while the target probabilities are used as the reference distribution P. The target probabilities are defined as the softmax of mean logits of the teacher models. The exact process is described in more details in [5]. As a comparative baseline, random partitions are also simulated, denoted as iid.

1) Noisy data: To simulate the fact that some user devices can be unreliable, the concept of noisy clients was tested. The training data on a noisy client had all labels replaced with randomly chosen classes, removing all information. Then, the performance was tested over the number of noisy clients $N_K \leq K$ to simulate erroneous or even adversarial clients.

H. Evaluation

For evaluation purposes, experiments were performed using FedAvg to explore the effects of four variables: The number of clients sampled (S), the class balance (α), the number of local epochs (E) and the number of noisy clients (N_K). Furthermore, the experiments regarding class balance and noisy clients were also performed using the FedDF algorithm.

The experiments varying local epochs were repeated on the physical federation of Raspberry Pi devices, both accessing the Internet through Ethernet and Wi-Fi. This was selected because altering this parameter changes the runtime of each communication round, while the behaviour for other tested parameters was approximately the same whether using the number of communication rounds or wall time as the x-axis. All experiments used the baseline listed in Table II except for the parameter being varied in each experiment. These were chosen based on existing literature, in particular works [4] [5], and some pilot experimentation.

TABLE II: Baseline parameters used for all experiments. Here, B refers to the training batch size.

K	S	α	E	L	N_K	B	η	γ
40	20	1	20	20	0	16	$5 \cdot 10^{-4}$	0.995

IV. Experimental Results

Figure 2 depicts the experiments conducted using the Pi system. Please note that these tests are time-restricted and, as a result, we have conducted a variety of communication cycles. The lines represent the average accuracy of three repetitions, whilst the shaded regions denote the best and worst repetitions at any given period. Additionally, the legend depicts the number of rounds (L) completed by each experiment before the 50-minute timeout. Figure 3 shows the varying clients per round over Ethernet and WiFi. A detailed analysis of the experimental results of the proposed method using the standard FedAvg algorithm is illustrated in Figure 4.

The results indicate that model convergence is possible even though data is distributed across devices; a conclusion substantiated by the fact that continuing the $E = 1$ training resulted in 65.3% of centralised learning accuracy, the p2p side-processing method reached a 73.4% and the p2p decentralized fl achieved 79.2% accuracy shown in Table III.

TABLE III: Test set accuracy when running the above learning algorithms until test set accuracy stopped improving for 3 steps. The used FedAvg algorithm used $E = 1$ and other baseline parameters. The centralised learning algorithm used the full training each epoch, but otherwise same optimization approach as FL.

Algorithm	**Steps before convergence**	**Final accuracy**
FedAvg	200 comm. rounds	62.1
Centralised	10 full epochs	65.3
P2P side processing	10 full epochs	73.4
P2P Decentralized FL	200 comm. rounds	79.2

From the experiments shown in Figure 2, we see that the majority of accuracy curves decline during the majority of the training phase. This impact becomes progressively severe as more local epochs pass. This troublesome behaviour is attributed to over-fitting. To verify this argument, test performances during local epochs are shown in Figure 8. During local epochs, the training accuracy on each client skyrockets, while the training accuracy plummets. The immediate countermeasure is to conduct a systematic hyperparameter search in which improved regularising methods are evaluated. A more general idea is to perform local early stopping, but designing such a rule is non-trivial as we observe instances of the global, average model improving even though all local models overfit, as observed in early communication rounds and during the less biased learning for $E = 10$, as displayed in Figure 5.

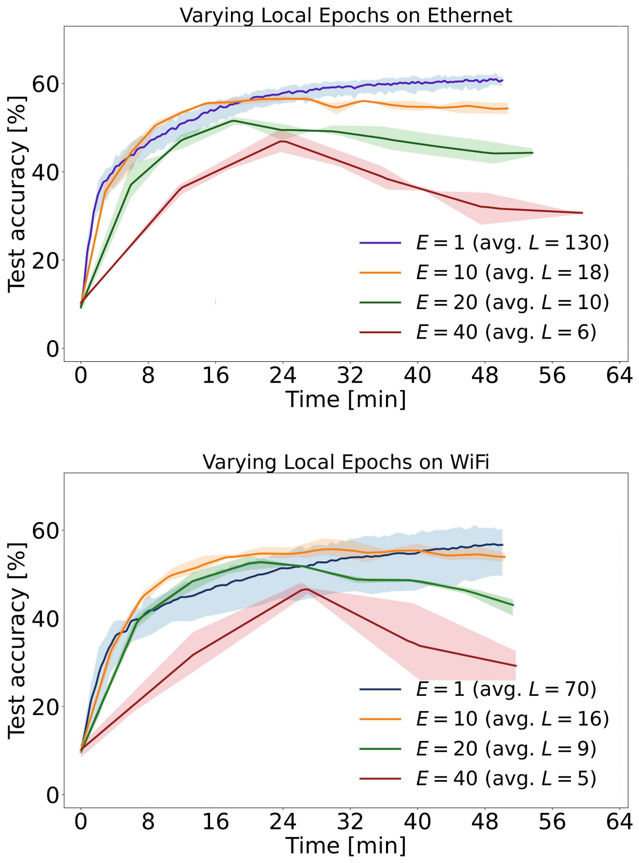


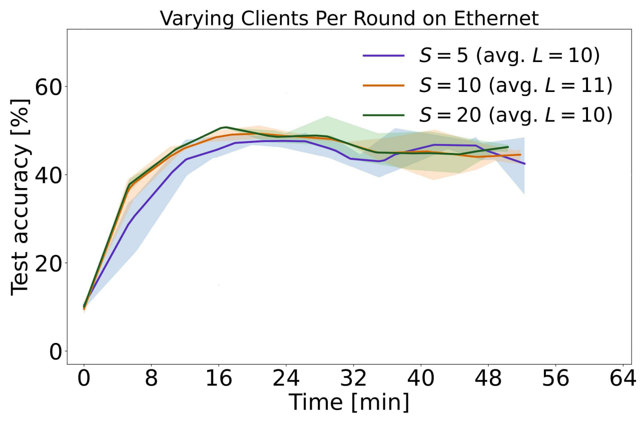
Fig. 2: Effect of the number of local epochs (E) using the Raspberry Pi setup on ethernet (top) and Wi-Fi (bottom).

V. Discussion

When concentrating on early learning that is less affected by overfitting, Raspberry Pi timing findings demonstrate that it is not always true that fewer local epochs are preferable. If training time is restricted, as demonstrated in 5 to 15 minutes for Ethernet and 5 to 40 minutes for Wi-Fi, the ideal number of local epochs may be increased.

During this period, the increased number of local epochs causes more time to be spent on training and less on communication, which explains why this impact is most pronounced on relatively sluggish Wi-Fi. Thus, there exists a tradeoff in which E should neither be too high to promote overfitting nor too low to slow down the pace at which training data is seen. When selecting this value, practitioners must consider available training time and system communication latencies.

As previously shown, when running for a set number of rounds, computational and communicational effects are neglected, and $E = 10$ seems to be the ideal balance between under- and overfitting. Even if datasets are relatively unbalanced, $\alpha = 1$, for the amount of clients sampled in each round S, FedAvg performs same whether sampling half of the clients or all of the clients. Even with just five clients every round, the overall accuracy was 93 %, demonstrating



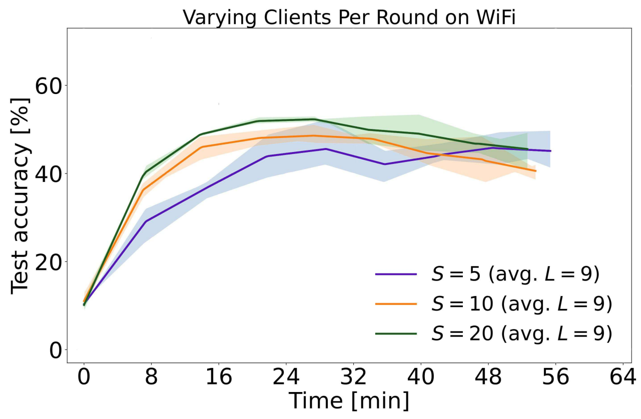
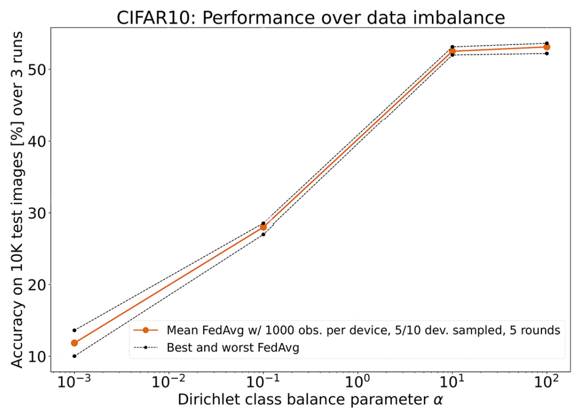


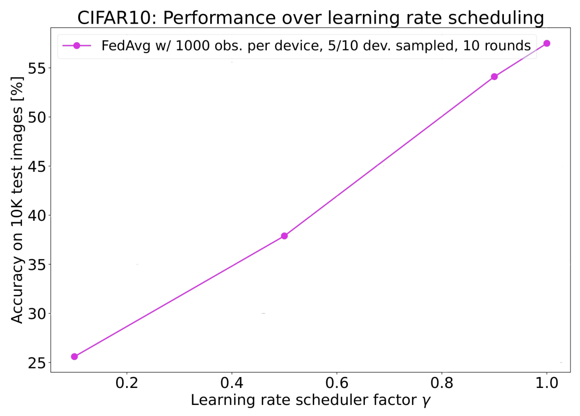
Fig. 3: Effect of the number of varying clients (S) using the Raspberry Pi setup on ethernet (top) and Wi-Fi (bottom).

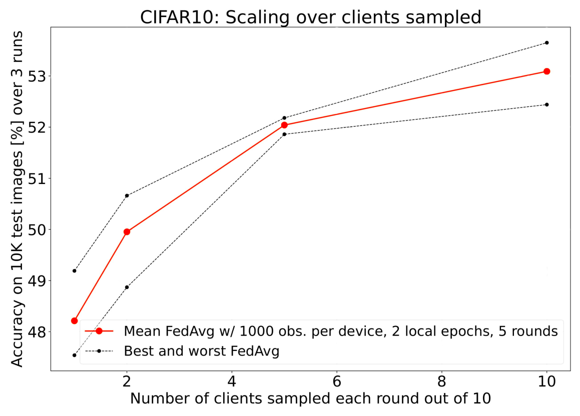
the consistency of model averaging. As seen in Figure 9, this disparity is greatest in the early rounds, when smaller values of S produce slower convergence. The lines represent the average performance of five repetitions, while the coloured regions denote the least and most efficient repetitions for each run. Additionally, Figure 6 depicts the changing alphas for the tests.

Class balancing findings demonstrate that decreasing α to 0.01 eliminated this stability. Training on $\alpha = 100$ or iid increased learning. FedDF trials using the same hyperparameters perform better. We hypothesise that distillation avoids the harmful effects of overfitted local models by fusing models without averaging over large, bias-inducing factors. Figure 10 shows that FedDF eliminates long-term decreasing performance. Empirical prediction probability distributions may be a better indicator of learnt knowledge than model weights.

This added robustness is strongly exemplified for FedDF in the noise experiments. Here, having 10 out of the 40 data partitions being noisy ruins the performance of FedAvg thoroughly, while FedDF can perform reasonably even at 30 out of 40. Yet again, averaging probabilities instead of model weights appears to minimise the negative impact of fusing models with heterogeneous parameter values.







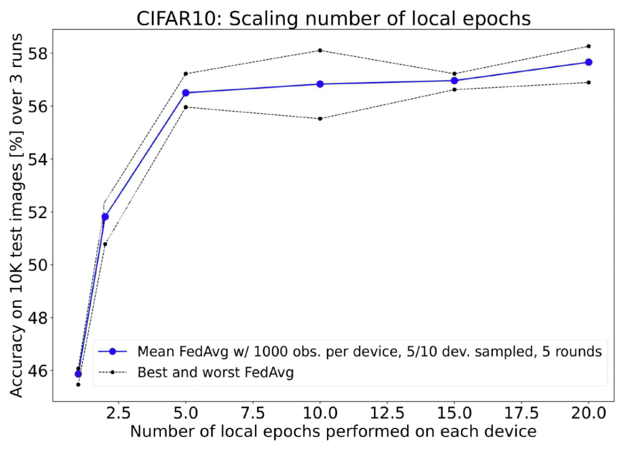


Fig. 4: Graphical Examination of Dirichlet Balance, Learning Rate, Local Epoches and Scalabilty.

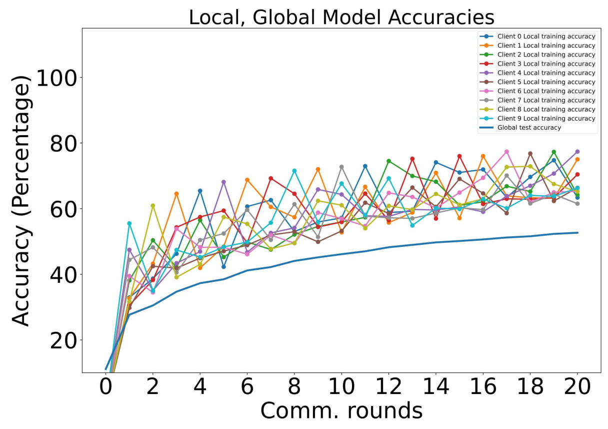


Fig. 5: Training trajectories when recording accuracies on each client in each communication round where each faint line corresponds to $E = 10$ local epochs of the sampled clients.

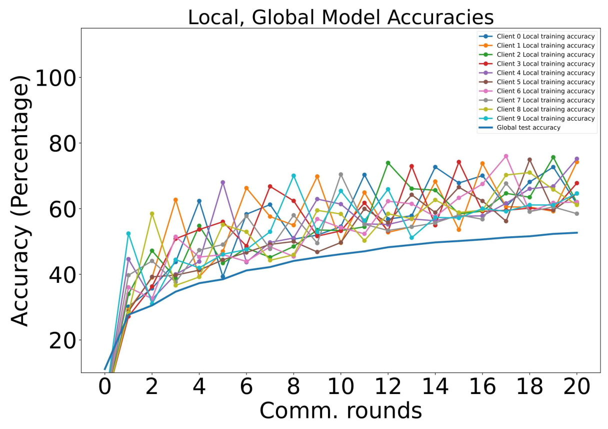


Fig. 8: Training trajectories when recording accuracies on each client in each communication round where each faint line corresponds to $E = 20$ local epochs of the sampled clients.

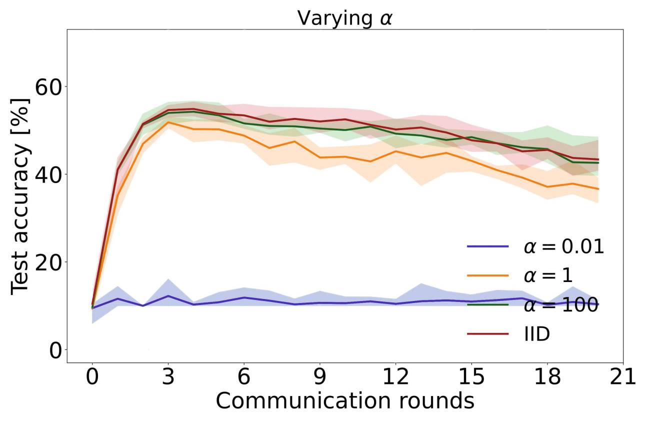


Fig. 6: Performance of the global model on the test at different communication rounds with varying number of class balance, α.

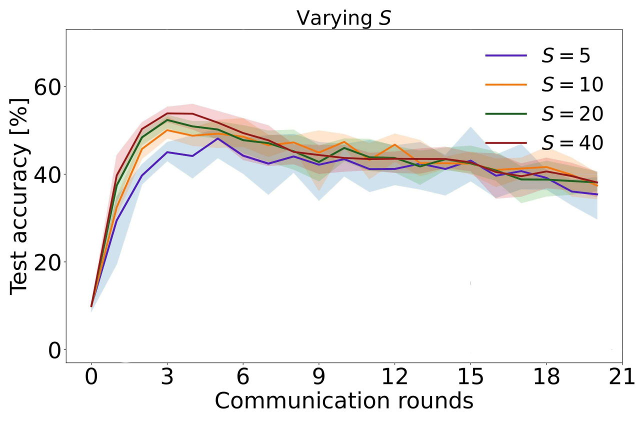


Fig. 9: Performance of the global model on the test at different communication rounds with varying number of clients sampled per round, S.

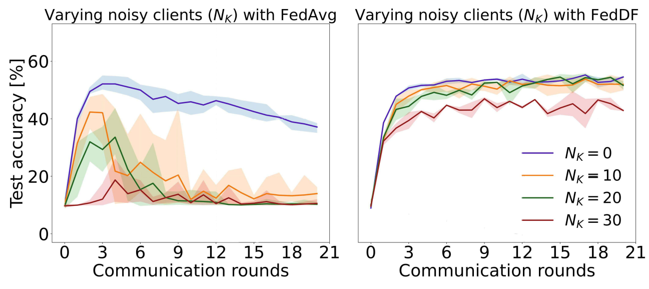


Fig. 7: Performance of the global model on the test set over communication rounds when varying noisy clients (N_k) using both FedAvg (left) and FedDF (right).

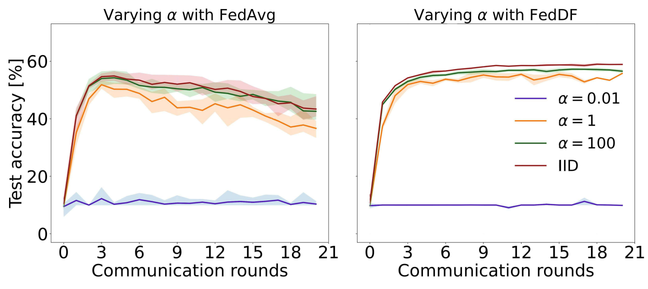


Fig. 10: Performance of the global model on the test set over communication rounds when varying data imbalance using both FedAvg (left) and FedDF (right).

VI. Conclusions and Future Work

In the context of this paper, the inner working of FedAvg and the FedDf algorithms were demonstrated along with the fact that privacy-preserving learning on physical devices is feasible as long as algorithmic choices are made with communication efficiency and robustness against data imbalance is taken in mind. Moreover, two proposed algorithms where the Federated Learning scenario can be transposed in a peer to peer environment were introduced. The experimental results indicate that using the simple yet very effective FedAvg algorithm and executing more local training steps on devices increased overfitting, while fewer epochs increased communication cost. FedDF, a distillation aggregation technique, minimised this overfitting and improved tolerance to varied data distributions at the expense of more central server processing. To sample each device a method for client-balancing Dirichlet sampling was initiated which ensures that each device is sampled as much times as its neighbors. Therefore, this method ensures that each chunk of the dataset which runs locally on every device, will be equally sampled across all devices providing equality among all participating devices. Hence, the overall dataset is not affected by features that might interfere with over-sampled labels.

Future directions of this work include the investigation of multiple realistic learning tasks, including computer vision models with higher performance, and a federation of devices with more computing power than the Raspberry Pi devices, such as smartphones. Another future aspect is investigate datasets that contain erroneous or missing values. Moreover, efficient sampling schemes as in [21]–[25] can be used to further improve the client-balancing method but this requires further investigation. Ultimately, the p2p scenario presented for side processing and P2P-FL is subject to future work along with [26] for possible fine-tuning optimizations.

References

[1] P. Kairouz, H. B. McMahan, B. Avent, A. Bellet, M. Bennis, A. N. Bhagoji, K. Bonawitz, Z. Charles, G. Cormode, R. Cummings, *et al.*, "Advances and open problems in federated learning," *Foundations and Trends® in Machine Learning*, vol. 14, no. 1–2, pp. 1–210, 2021.

[2] P. Kairouz, "Advances and open problems in federated learning," *Foundations and Trends® in Machine Learning*, vol. 14, no. 1–2, pp. 1–210, 2021.

[3] Q. Yang, Y. Liu, T. Chen, and Y. Tong, "Federated machine learning: Concept and applications," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 10, no. 2, pp. 1–19, 2019.

[4] 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*, pp. 1273–1282, PMLR, 2017.

[5] T. Lin, L. Kong, S. U. Stich, and M. Jaggi, "Ensemble distillation for robust model fusion in federated learning," *Advances in Neural Information Processing Systems*, vol. 33, pp. 2351–2363, 2020.

[6] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, M. Ranzato, A. Senior, P. Tucker, K. Yang, *et al.*, "Large scale distributed deep networks," *Advances in neural information processing systems*, vol. 25, 2012.

[7] H. B. McMahan, D. Ramage, K. Talwar, and L. Zhang, "Learning differentially private recurrent language models," *arXiv preprint arXiv:1710.06963*, 2017.

[8] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth, "Practical secure aggregation for privacy-preserving machine learning," in *proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pp. 1175–1191, 2017.

[9] Y. Aono, T. Hayashi, L. Wang, S. Moriai, *et al.*, "Privacy-preserving deep learning via additively homomorphic encryption," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 5, pp. 1333–1345, 2017.

[10] Y.-R. Chen, A. Rezapour, and W.-G. Tzeng, "Privacy-preserving ridge regression on distributed data," *Information Sciences*, vol. 451, pp. 34–49, 2018.

[11] H. Kim, J. Park, M. Bennis, and S.-L. Kim, "On-device federated learning via blockchain and its latency analysis," *arXiv preprint arXiv:1808.03949*, 2018.

[12] V. Smith, C.-K. Chiang, M. Sanjabi, and A. S. Talwalkar, "Federated multi-task learning," *Advances in neural information processing systems*, vol. 30, 2017.

[13] W. Du, Y. S. Han, and S. Chen, "Privacy-preserving multivariate statistical analysis: Linear regression and classification," in *Proceedings of the 2004 SIAM international conference on data mining*, pp. 222–233, SIAM, 2004.

[14] W. Du and M. J. Atallah, "Privacy-preserving cooperative statistical analysis," in *Seventeenth Annual Computer Security Applications Conference*, pp. 102–110, IEEE, 2001.

[15] L. Wan, W. K. Ng, S. Han, and V. C. Lee, "Privacy-preservation for gradient descent methods," in *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 775–783, 2007.

[16] A. F. Karr, X. Lin, A. P. Sanil, and J. P. Reiter, "Privacy-preserving analysis of vertically partitioned data using secure matrix products," *Journal of Official Statistics*, vol. 25, no. 1, p. 125, 2009.

[17] J. Vaidya and C. Clifton, "Privacy preserving association rule mining in vertically partitioned data," in *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 639–644, 2002.

[18] K. Cheng, T. Fan, Y. Jin, Y. Liu, T. Chen, D. Papadopoulos, and Q. Yang, "Secureboost: A lossless federated learning framework," *IEEE Intelligent Systems*, vol. 36, no. 6, pp. 87–98, 2021.

[19] S. Hardy, W. Henecka, H. Ivey-Law, R. Nock, G. Patrini, G. Smith, and B. Thorne, "Private federated learning on vertically partitioned data via entity resolution and additively homomorphic encryption," *arXiv preprint arXiv:1711.10677*, 2017.

[20] B. Schoenmakers and P. Tuyls, "Efficient binary conversion for Paillier encrypted values," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 522–537, Springer, 2006.

[21] C. Karras, A. Karras, and S. Sioutas, "Pattern recognition and event detection on iot data-streams," *arXiv preprint arXiv:2203.01114*, 2022.

[22] C. Karras and A. Karras, "Dbsop: An efficient heuristic for speedy mcmc sampling on polytopes," *arXiv preprint arXiv:2203.10916*, 2022.

[23] C. Karras, A. Karras, M. Avlonitis, and S. Sioutas, "An overview of mcmc methods: From theory to applications," in *Artificial Intelligence Applications and Innovations. AIAI 2022 IFIP WG 12.5 International Workshops* (I. Maglogiannis, L. Iliadis, J. Macintyre, and P. Cortez, eds.), (Cham), pp. 319–332, Springer International Publishing, 2022.

[24] C. Karras, A. Karras, M. Avlonitis, I. Giannoukou, and S. Sioutas, "Maximum likelihood estimators on mcmc sampling algorithms for decision making," in *Artificial Intelligence Applications and Innovations. AIAI 2022 IFIP WG 12.5 International Workshops* (I. Maglogiannis, L. Iliadis, J. Macintyre, and P. Cortez, eds.), (Cham), pp. 345–356, Springer International Publishing, 2022.

[25] C. Karras, A. Karras, G. Drakopoulos, K. Tsakalidis, P. Mylonas, and S. Sioutas, "Weighted reservoir sampling on evolving streams: A sampling algorithmic framework for stream event identification," in *Proceedings of the 12th Hellenic Conference on Artificial Intelligence*, SETN '22, (New York, NY, USA), Association for Computing Machinery, 2022.

[26] A. Karras, C. Karras, K. C. Giotopoulos, I. Giannoukou, D. Tsolis, and S. Sioutas, "Download speed optimization in p2p networks using decision making and adaptive learning," in *Proceedings of the ICR'22 International Conference on Innovations in Computing Research* (K. Daimi and A. Al Sadoon, eds.), (Cham), pp. 225–238, Springer International Publishing, 2022.