Future Generation Computer Systems 162 (2025) 107475

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

# Network-aware federated neural architecture search

Göktuğ Öcal *, Atay Özgövde

*Boğaziçi University, Bebek, 34342, Istanbul, Turkiye*

ARTICLE INFO

*Keywords:*
Neural architecture search
Federated learning
Network pruning
Client selection
Client grouping
Network emulation

ABSTRACT

The cooperation between Deep Learning (DL) and edge devices has further advanced technological developments, allowing smart devices to serve as both data sources and endpoints for DL-powered applications. However, the success of DL relies on optimal Deep Neural Network (DNN) architectures, and manually developing such systems requires extensive expertise and time. Neural Architecture Search (NAS) has emerged to automate the search for the best-performing neural architectures. Meanwhile, Federated Learning (FL) addresses data privacy concerns by enabling collaborative model development without exchanging the private data of clients.

In a FL system, network limitations can lead to biased model training, slower convergence, and increased communication overhead. On the other hand, traditional DNN architecture design, emphasizing validation accuracy, often overlooks computational efficiency and size constraints of edge devices. This research aims to develop a comprehensive framework that effectively balances trade-offs between model performance, communication efficiency, and the incorporation of FL into an iterative NAS algorithm. This framework aims to overcome challenges by addressing the specific requirements of FL, optimizing DNNs through NAS, and ensuring computational efficiency while considering the network constraints of edge devices.

To address these challenges, we introduce Network-Aware Federated Neural Architecture Search (NAFNAS), an open-source federated neural network pruning framework with network emulation support. Through comprehensive testing, we demonstrate the feasibility of our approach, efficiently reducing DNN size and mitigating communication challenges. Additionally, we propose Network and Distribution Aware Client Grouping (NetDAG), a novel client grouping algorithm tailored for FL with diverse DNN architectures, considerably enhancing efficiency of communication rounds and update balance.

## 1. Introduction

Deep learning (DL) has enabled remarkable progress which inevitably revolutionized our daily lives and the services people are using.

That remarkable progress cannot be thought of without the effect of collaboration of DL and edge devices which are computing devices located at the periphery of a network, acting as data sources. Integration of DL on edge devices enabled advanced real-time intelligent applications at the edge of the network by processing the data generated by these devices. Processing the raw data and running DL models on the edge reduces latency compared to sending data to centralized cloud servers, loads on the server, and optimizes bandwidth usage.

Edge devices are highly crucial for DL-powered real-time applications that demand immediate insights such as autonomous vehicles, healthcare monitoring, and industrial automation. Autonomous vehicles, for instance, rely on DL models to process sensor data in real-time for safe navigation and decision-making [1]. Similarly, healthcare monitoring systems utilize DL to continuously analyze patient data from wearable devices, detecting anomalies and enabling timely medical interventions [2]. Smart city applications [3] also depend on DL to manage traffic, energy distribution, and environmental monitoring through IoT sensors, requiring immediate data processing for efficient urban management. These examples underscore the importance of edge computing in delivering the low-latency processing necessary for effective time-sensitive DL applications.

In addition to the requirements of real-time applications deployed on edge devices, keeping and processing data on edge devices ensures privacy and security. On the other hand, local data of the edge devices may reduce the generalization performance of DL models by causing underfitting or bias. In that sense, the local data of the edge devices is limited when compared with the amount of data in the centralized cloud servers, as expected. To preserve DL capabilities while overcoming data privacy issues, Federated Learning (FL) has emerged. Federated learning is a Machine Learning (ML) approach

* Corresponding author.
*E-mail addresses:* goktug.ocal@std.bogazici.edu.tr (G. Öcal), ozgovde@bogazici.edu.tr (A. Özgövde).

https://doi.org/10.1016/j.future.2024.07.053
Received 21 February 2024; Received in revised form 15 July 2024; Accepted 29 July 2024
Available online 8 August 2024

that ensures data privacy and mitigation of computational overhead on cloud servers through the contribution of several clients to develop a shared, robust machine learning model without exchanging data. When compared to centralized training on persisting data, federated learning offers clear privacy advantages because of its distributed computing nature [4]. In an FL environment, all the communication between clients, data centers, and servers happens over the network. Therefore, from an unusual aspect, network becomes another factor affecting the performance of DL system besides other factors such as data quality and neural network architecture.

Edge devices suffer from their resource-constrained characteristic. The computation-intensive nature of DL model architectures causes a substantial problem when deploying these models on edge devices. In such a case, DL architectures that balance prediction performance and computational efficiency are required. However, the design of DNNs involves various considerations such as network architecture, weights, and hyper-parameters, all of which significantly impact performance. Moreover, developing DNN architectures manually requires extensive expertise in DL due to multiple reasons [5]. The extensive search space includes a multitude of options regarding the number and types of layers, connections, and hyperparameters, requiring comprehensive understanding for effective exploration. Customizing architectures for specific tasks is labor-intensive, often resulting in suboptimal designs. The design process demands meticulous consideration of various parameters, such as layer types, operations, activation functions, training data, and deployment constraints, which necessitates specialized engineering skills. To tackle those problems, Neural Architecture Search (NAS) is coined by Zoph et al. in [6]. Automating the search of various DNN architectures and configurations, NAS seeks to find models that are efficient and best-performing.

While DL-powered services require real-time decision-making, edge devices generally have a limited computational budget (latency, memory, FLOPs, etc.) and especially changing network conditions (bandwidth, delay, jitter, location, etc.) [7]. That situation increases the importance of computational efficiency and data privacy while maintaining the model performance for use cases, which are shaping the ongoing evolution of deep learning applications in our interconnected and data-driven world. In that sense, edge devices can leverage two different concepts: FL for data privacy and distributing the computing overhead to multiple devices, and NAS for finding computing-efficient DNN architectures that can fit the resource budget limitations of edge devices. The cooperation of those two approaches makes edge devices able to preserve their local data and run real-time applications properly.

The trend in DNN design is towards larger models for better accuracy, but their complexity hinders low inference latency [8]. Following this trend, larger and deeper neural networks with custom-designed architectures have been created [9]. Regarding the trade-off between the validation accuracy and the computational cost, practitioners usually consider only accuracy since the research community and industry benchmarks traditionally emphasize validation accuracy. Even though traditional DNN architecture design has the objective of increasing validation accuracy when the computational and network limitations of edge devices are considered, other objectives such as computational cost and size of the DNNs become relevant. It is very challenging for practitioners to achieve these objectives simultaneously; therefore, neural network pruning and efficient neural architecture search methods have been developed to find the most feasible DNNs for heterogeneous devices. In this context, efficiency is related to the minimization of the computational cost and the inference latency of the models produced by NAS algorithms.

On the FL side, distributed training on decentralized devices requires robust and efficient communication, making network conditions a critical factor in terms of both training speed and model performance. Clients with limited bandwidth or high latency cause longer upload/download durations, leading to slower training convergence and increased communication overhead. Furthermore, network variety can cause fairness issues, as clients with poor connections struggle to contribute their data effectively, potentially resulting in models biased towards those with better network conditions. To reduce communication overhead, selecting a set of clients for FL tends to cause biased results because of varying data distributions of clients. Missing updates, caused by communication delays, introduce noise into the training process, that prevents the model performance and convergence.

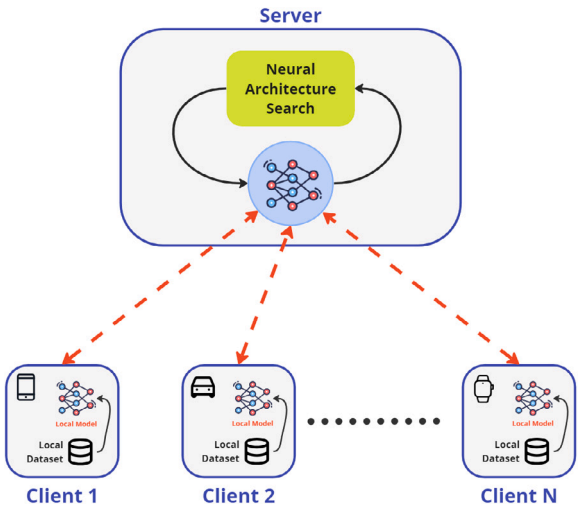



**Fig. 1.** Neural Architecture Search in a federated scenario. NAS Algorithm is executed in the center with clients contributing their data in a privacy-preserving manner.

Even though federated NAS and federated network pruning algorithms and systems help find successful lightweight models, such algorithms with tremendous computation costs cause communication and computation costs on clients. Also, the network conditions of edge and server devices have effects on that federated process. There have been various researches in efficient NAS and FL domains but to the best of our knowledge, those concepts are not handled from a detailed network perspective. In this research, we have studied the behaviors of the devices that have been involved in the FL process with a highly capable and realistic framework as illustrated in Fig. 1. Our focus is on the effects of network conditions federated neural network pruning systems and possible solutions for computation cost, communication cost, and convergence time. Moreover, we have set up a fully open-source federated neural network pruning system. Thus, main contributions of our study can be summarized as:

- Our first significant contribution is the development of an open-source federated neural network pruning framework with network emulation support. The framework, named Network-Aware Federated Neural Architecture Search (NAFNAS)[1], is specifically built to address the challenges associated with model pruning in a distributed network of clients within federated learning settings and various network characteristics. By efficiently reducing the size of neural networks, our framework tackles the complexities of transmitting and updating large DNN models with different architectures on resource-constrained devices. Through rigorous testing and experimentation, we have demonstrated the feasibility of our federated neural network pruning approach while maintaining optimal model performance.
- The second key contribution is a novel client grouping algorithm, called Network and Distribution Aware Client Grouping (NetDAG), tailored for federated learning in an environment of models with different DNN architectures. This algorithm has been

[1] https://github.com/goktugocal/NAFNAS

developed to account for both the non-identically distributed (non-IID) characteristic of the local data of the client and the limitations caused by network bandwidth constraints. By optimizing the placement of clients into groups based on their class distributions and network bandwidth tiers, our algorithm improves the efficiency in communication round duration and balance of federated learning updates.
- Our third main contribution is an in-depth study of how network bandwidth affects federated learning, taking particular interest in the length of communication rounds. After conducting comprehensive testing, we revealed the connection between changes in network bandwidth and the effectiveness of federated learning processes. To mitigate the consequences of low network bandwidth, this research offers useful insights that enable the design of more resilient federated learning systems.

**Organization.** The rest of this paper is organized as follows. In Section 2, crucial concepts in this paper are explained briefly. Section 3 gives the prior works through the Federated Neural Architecture Search. We have explained our proposed framework and client grouping algorithm in detail in Section 4. Section 5 includes all the experiments and discussions in our work from NAS, FL, and network perspectives. Finally, Section 6 summarizes our work and contributions and highlights the key results.

## 2. Background

### 2.1. Neural architecture search and neural network pruning

Traditional approaches to developing strong models from a dataset require intuition, experience, and time-consuming manual fine-tuning of algorithmic parameters. Constructing a well-performing machine learning pipeline is often tedious and error-prone for scientists or developers [10]. Neural Architecture Search (NAS), a subfield of AutoML, automates the design of deep learning (DL) architectures, finding the most suitable ones for a given task.

NAS involves selecting and optimizing various architectural components such as layers, layer sizes, and activation functions. The process builds a *search space* of candidate architectures and identifies the best-performing one, typically in terms of accuracy. Search strategies include black-box methods like Bayesian optimization, reinforcement learning, evolutionary search, and one-shot approaches like supernet and hypernet-based methods. Performance estimation techniques evaluate architectures considering factors like model size and inference duration to address cost and device limitations.

NAS uses a single-objective optimization approach to maximize accuracy, that generally leads to the creation of complex and resource-intensive models. Multi-objective optimization in NAS, on the other hand, seeks a compromise between computational efficiency and accuracy by taking into account various parameters like FLOPs, model size, memory utilization, inference latency, and power consumption. This approach adjusts search strategies to account for multiple criteria, identifying a Pareto front of architectures that represent optimal trade-offs. The advantages of multi-objective optimization include developing models that are both highly accurate and efficient, making them suitable for deployment on resource-constrained devices.

The breakthrough performance of deep neural networks (DNNs) relies on a large number of parameters and significant computational power [11]. However, state-of-the-art DNN models require resources that exceed those available on many edge and embedded devices. Network pruning reduces model complexity while retaining accuracy by removing redundant connections and weights, creating lighter models suitable for resource-limited systems. Pruning techniques, guided by criteria like feature shift minimization and matrix sparsification, effectively reduce overparameterized models without significantly sacrificing accuracy.

There are two common approaches in neural network pruning:

- **Non-structured pruning** zeroizes unnecessary connections or weights.
- **Structured pruning** removes entire components, such as filters in CNNs, to reduce size and computational cost without affecting architectural stability.

Neural network pruning benefits include smaller models that are ideal for deployment on resource-constrained platforms like mobile phones or embedded systems, faster prediction times crucial for real-time applications, and lower energy consumption. Pruning can be tailored to specific hardware or software platforms, ensuring models fit within device memory constraints.

Enhancing model efficiency through neural network pruning often leads to lower performance because of the balance needed between reducing parameters and keeping the model's ability to learn well. Pruning removes weights, neurons, or filters that are important for recognizing complex patterns in the data, which decreases the model's ability to represent these patterns and results in the loss of important information. While pruning can help prevent overfitting by getting rid of unnecessary parameters, it can also cause underfitting if too many parameters are removed, making the model too simple to capture the data accurately. This reduction in complexity can harm the model's ability to generalize from the training data to new data. Therefore, pruning must be done carefully to avoid a significant drop in the model's accuracy.

There are several approaches to the resource constraint neural network pruning problem, one of them is called NetAdapt [12]. NetAdapt is a structured pruning algorithm that automatically simplifies a pre-trained deep neural network to meet the resource budget of a platform while maximizing accuracy. It incorporates direct metrics, such as latency and energy consumption, and therefore does not require platform-specific information. Then, it enables a resource reduction schedule:

$$\begin{aligned} & f_i^* = \arg\min_{f_i} Cost(f_i) \\ & \text{s.t. } C_j(f_i) \le C_j(f_{i-1}) - \Delta R_{i,j} \end{aligned} \tag{1}$$

where the objective is finding $f_i^*$ at iteration $i$th. $f_i$ is the network generated in $i$th iteration, and $f_0$ is the pretrained DNN model. The current resource budget denoted as $C_j(f_{i-1}) - \Delta R_{i,j}$ becomes tighter when the iterations increase. $R_{i,j}$ defines how much the constraint becomes tighter for $j$th resource at $i$th iteration.

The NetAdapt algorithm selects convolutional layers (CONV) and fully connected layers (FC) to fit in the resource constraint. The algorithm consists of 3 steps in pruning iterations; (1) choosing the number of filters, (2) choosing which filters to preserve, (3) short-term fine tuning; illustrated in Fig. 2.

The first phase entails determining the optimal number of filters to preserve in a specific layer through empirical measurements. Each layer of the network is investigated individually and there is a gradual reduction in the number of filters within the target layer. Reduction is decided by an evaluation of the resource utilization of simplified networks and the candidate child architecture with a maximum size of filters is selected. In the filter selection phase, it is decided which filters to retain, building on the architecture determined in the previous step. To keep things straightforward, a magnitude-based method is applied. Specifically, $N$ filters with the highest $\ell 2$-norm magnitude are kept, where $N$ corresponds to the number of filters established in the preceding step. The fine-tuning process in NetAdapt consists of short-term and long-term phases, during each iteration, the pruned networks undergo a relatively brief fine-tuning process, with the aim of restoring accuracy. In the measurement of resources such as inference latency, the algorithm generates a latency lookup table beforehand by measuring the inference latency of different layers with their sizes. In the pruning process, the algorithm just checks the look-up table whether the resource of the pruned model meets the constraint.

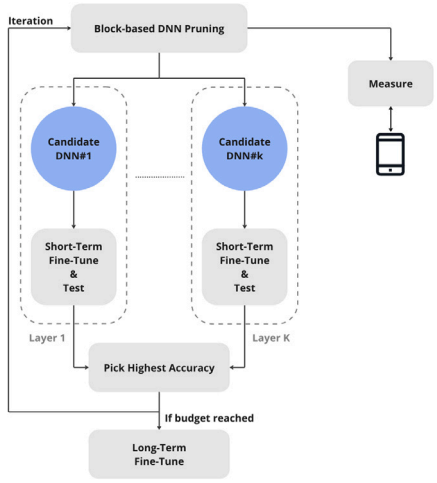



**Fig. 2.** Algorithm flow of NetAdapt.

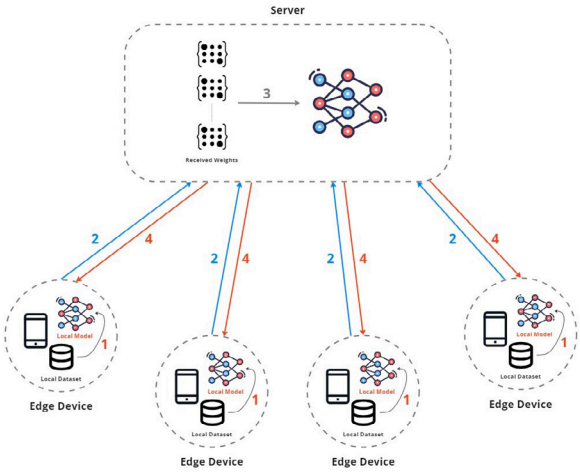



**Fig. 3.** Steps of Federated Learning.

### 2.2. Federated learning

To maintain data privacy, in 2016, Google proposed the Federated Learning solution [13] which spreads the data management and model training processes to individual clients or participants.

*Federated learning* represents a machine learning methodology wherein the model undergoes training across numerous decentralized devices or servers, each retaining its local data samples without the need for data exchange. The server coordinates the federated learning process for training and evaluation steps and gives required commands to the clients. A round is defined, and illustrated in Fig. 3, as the (1) local training process that a client trains a DNN model with local dataset, (2) sending updated weights to the server, (3) aggregating received weights with a proper aggregation algorithm, therefore server waits for each participant client to collect sent weights, (4) distributing the new global model to the clients for next round. Another step can be sending the new global model to clients again for the evaluation step and collecting the performance measurements such as accuracy, latency, etc. It is crystal clear to observe that in each round there are huge amount of communication between the server and clients due to the size of model weights. This approach prioritizes privacy and security by conducting the training process on a per-user basis, eliminating the need to share data with the server.

"Heterogeneous clients" in a federated learning setting are users with different network properties such as network bandwidth. Clients who can participate more frequently and produce better results can be given priority by the server, which could introduce bias into the federated learning process. This undesirable state might result in performance losses, showing how crucial it is to create techniques that preserve clients from bias in a federated learning environment with a range of limitations.

### 2.3. IID and Non-IID data distributions

IID (Independent and Identically Distributed) refers to a characteristic of a dataset where each data point is independent of others, and all data points are created from the same underlying probability distribution. This assumption simplifies statistical analyzes and is commonly assumed in traditional machine learning settings. On the other hand, Non-IID (Non-Identically Distributed) characterizes a dataset where data points may have different probability distributions, variability, or heterogeneity in the statistical properties. To give an example, Li et al. [14] stated that non-IID scenario, often arises due to diverse scanner vendors, patient demographics, and varying disease characteristics across institutions. Non-IID scenarios are closer, compared with IID data, to real world scenarios, such as federated learning or datasets collected from diverse sources.

The Non-IID dataset causes the model weights trained on one subset of the data to differ significantly from those learned on another subset, which is called *weight divergence*. In the context of federated learning, local datasets of clients with unique statistical properties would cause the weights to diverge differently therefore that situation would prevent the weight aggregation algorithm from converging or generalizing [15], that situation raises a challenge.

**Dirichlet distribution** is employed to model a situation where datasets are distributed unevenly among agents. This is achieved by assigning samples within each class unevenly across agents, using proportions sampled from a symmetrical Dirichlet distribution $Dir(\alpha)$. The $\alpha$ parameter is tunable to regulate the level of uneven distribution. Higher $\alpha$ values result in more balanced distributions, while lower values lead to more uneven splits [16].

**Earth Mover's Distance (EMD)**, also known as Wasserstein distance, is a metric that measures the similarity of two distributions defined in the same space. The fundamental approach in EMD is determining the minimum work required to transform one distribution into another. EMD is formulated as;

$$EMD_k = \sum_{i=1}^{C} \|p_{y=i}^{(k)} - P_{y=i}\| \tag{2}$$

where $C$ denotes the number of class types, $p_{y=i}^{(k)}$ refers to the proportion of the number of the $i$th type data of client $k$ and $p_{y=i}$ refers to the global proportion. It is already known that higher EMD values across the clients lead to a higher weight divergence between locally trained models and lower validation accuracy on the aggregated global model [15]. Authors have proposed a data-sharing strategy from server to client to balance the distributions but that solution requires a common dataset on the server.

## 3. Related work

### 3.1. Neural architecture search

Neural Architecture Search is the process of building the best-performing neural network architecture for a given task in an automated way. NAS has already shown success in various tasks [17–19]. One of the prior works [6] by Google Brain utilized Reinforcement Learning (RL) in the neural architecture search process. The proposed method uses Recurrent Neural Networks (RNN) for architecture construction and RL to maximize the accuracy of a validation dataset, where the controller in RL learns expected validation errors and proposes better architecture in the following iterations. The proposed

method has been tested on CIFAR-10 and Penn Treebank datasets and used 800 GPUs for two weeks to achieve SOTA results. By realizing there are certain motifs in architecture engineering Zoph et al. [20] defined two types of convolutional cells for architecture building; one returns a feature map in the same dimension other reduces the dimension. That approach costs 500 GPUs and 4 days. ENAS (Efficient-NAS) [21] used a similar approach but used weight sharing to speed up the reward estimation. The search space has been effectively reduced by sharing weights across different architectures, which also allows faster exploration and evaluation of various architectures. ENAS method achieved similar results with only 1 GPU and less than 16 h.

Early, it was a common approach to use evolutionary algorithms for optimizing the neural network architecture [22,23]. A population of architectures is updated iteratively by evolutionary NAS methods. Each phase involves selecting one or more parent architectures from the population by its validation accuracy, combining them, and undergoing mutations to produce new child architectures [24]. Real et al. [25] randomly sampled the architectures from the search space and also used regularized evolution that eliminates the architecture in each phase which has been in the population for a long time. On the other hand, Elsken et al. [26] selected the best-performing architectures as parents from the population. LEMONADE [27] is a multi-objective evolutionary NAS algorithm that aims to minimize multiple objectives by considering the trade-off between predicting accuracy and consumption of resources.

RL and evolution-based NAS algorithms iteratively find architectures from search space and train them to obtain their performance. However, these approaches require immense computational costs because of the process of training thousands of architectures individually. With one-shot techniques each architecture in the search space is not trained from zero, just the opposite all architectures are trained by single training of a *supernetwork*.

A supernetwork is a large network that can build all possible neural network architectures in the search space by containing every possible operation [24,28]. The supernetwork idea is introduced by Saxena et al. [29]. The method proposed by Bender et al. [28], first trains the supernetwork and extracts the best sub-networks from the trained supernetwork. Once-for-all [30] is proposed to find an efficient neural network architecture to fit in memory and inference latency requirements. Once-for-all uses the progressive shrinking approach to generate neural network architectures with different sizes.

The most popular one in supernetwork-based NAS methods is DARTS (Differentiable Architecture Search) [31] proposed by Liu et al. The DARTS method uses continuous relaxation to represent the search space which makes it possible to use the gradient descent algorithm to extract best performing architecture more rapidly.

### 3.2. Neural network pruning

Convolutional Neural Network (CNN) is a popular branch of DNNs, especially in image processing tasks. Usually, CNNs are over-parametrized [32]. Advancements in the last decade in DL and DNNs and their use cases in edge devices have brought many kinds of research in neural network pruning along. There are various categories in these researches [33]. Magnitude-based pruning methods [34] measure the degree of importance of weights and neurons by their magnitudes. Sensitivity analysis-based pruning [35] investigates the effect of weights on the validation loss by removing or perturbing them. Pruning with knowledge distillation [36] is another approach of neural network pruning where the main model, that is to be pruned, is teaching the more efficient pruned model in the training phase. Quantization methods [37] focuses on quantization, binary, and low-precision representation of the DNN model for compression. Also, NAS-based methods [38] have been used to find more compact and efficient DNN models.

A comprehensive pruning method was proposed by Lin et al. [35] to remove unnecessary channels from a deep neural network. Their approach consists of two steps: first, the significance of each channel was measured, which resulted in the removal of unnecessary channels from all layers. Second, the method dynamically modifies filter accuracies by comparing sparse and pruned networks to fix any improperly pruned channels. To accelerate the inference process, Li et al. [39] proposed a refined pruning mechanism where channel-wise significance indexes and layer-wise sparsity of convolutional layers have been focused. Another structured pruning algorithm proposed by Chung et al. [40] where the proposed method prunes specific convolution channels in the first layer of CNN because of the fact that pruning of the first layer lets compressing the rest of the convolution layers successfully.

The foundation of unstructured pruning was a heuristic method for eliminating unnecessary parameters [41]. Han et al. [42] introduced the train-prune-retrain method to detect important connections to reduce the storage and computational cost of DNN. Yang et al. [43] utilized the energy cost of each layer in order to select and order layers to prune. NetAdapt [12] is another approach in neural network pruning that optimizes the pruning process by considering another objective such as inference latency and computational cost constraints.

PDAS [44] is an automated pruning approach that combines differentiable searching and evolutionary updating, effectively expanding the search space while maintaining search efficiency. This results in a pruned network with superior computational efficiency, as demonstrated by comprehensive experiments showing better accuracy and reduced computational costs compared to other methods, particularly for deeper or wider neural networks.

Quantization of DNN models is a process of reducing the precision of the numerical values used to represent the weights and parameters of the network. Post-training quantization is a process where quantization of weights of the DNN model, followed by re-optimization to produce a quantized model with scales [45]. Quantization-aware training is a technique to train DNN with the awareness that the model is later to be quantized to lower precision; to integer values, typically 8-bits [46,47].

KCD [48] (knowledge condensation distillation) focuses on knowledge distillation techniques and demonstrates significant improvements in accuracy across various teacher-student pairs, showcasing the effectiveness of condensing knowledge for efficient learning.

### 3.3. Federated learning

Federated Learning (FL), introduced by McMahan et al. [49], is a distributed machine learning (ML) framework designed to preserve data privacy. It is used in various fields such as mobile keyboard prediction [50], financial fraud detection [51], and precision medicine [52]. However, FL faces challenges including privacy protection, statistical heterogeneity, device heterogeneity, and communication heterogeneity.

ML techniques generally assume data are independent and identically distributed (IID) [53], but non-IID data distributions pose a problem of data heterogeneity. Hsieh et al. [54] investigated the effect of the degree of data skew on the performance of DNN models. FedAvg [49] utilizes the stochastic gradient descent (SGD) algorithm with a global aggregation setting, but it is insufficient for non-IID data [15]. Methods to address this include sharing nonprivate data [55], generating missing samples via GANs [56], dynamic sample selection [57], and client selection based on class distribution [58].

Optimization techniques include FedSA [59], which uses a semi-asynchronous mechanism, and adaptive optimizers like ADAGRAD, ADAM, and YOGI integrated into FedAvg [60]. FedProx [61] adds a proximal term to the local optimization objective to balance local and global weights in non-IID conditions.

Despite FL's aim to ensure data privacy, communication over networks can introduce vulnerabilities [62]. Differential privacy strategies, such as DP-FedAvg [63], prevent attackers from inferring local

datasets. [64] applied the DP-FedAvg algorithm to mobile devices. FedMD-NFDP [65] employs a noise-free differential privacy mechanism in a knowledge distillation-based FL method. Bonawitz et al. [66] developed a scalable federated learning system for mobile devices, utilizing Secure Aggregation and differential privacy techniques to ensure data confidentiality. Their system addresses challenges like device dropouts and synchronization overhead, providing practical solutions for large-scale deployments. A differential private federated transfer learning architecture for mental health monitoring has been presented in a recent work [67]. This approach addresses both data insufficiency and privacy concerns while achieving notable improvements in accuracy and recall for stress detection by combining federated learning with differential privacy and transfer learning.

To address data heterogeneity, FedBnR [68] breaks down skewed tasks into unbiased subtasks, achieving superior computational efficiency and accuracy compared to other personalized FL algorithms. Knowledge distillation methods, like FedMD [69], reduce communication costs and handle model heterogeneity by sharing logits instead of model parameters. FedDistill [70] and FedKD [71] propose federated distillation methods that maintain data privacy while training global models.

These approaches collectively aim to improve the performance and robustness of FL in the face of data and device heterogeneity while ensuring privacy and reducing communication costs.

### 3.4. Federated neural architecture search

The demand for privacy-preserving distributed training increased interest in federated NAS. Federated NAS methods can be categorized as offline and online Federated NAS [72]. In offline Federated NAS, the architecture search process is conducted in a centralized server and the candidate DNN architecture is distributed to the participant clients for local training and deployment. Zhu et al. [73] proposed a genuine offline federated NAS framework where an evolutionary optimization algorithm for selecting optimal architecture from the Pareto front. On the other hand, in online federated NAS, the search for the optimum of the DNN architecture and the distribution of the DNN is performed simultaneously, the candidate architecture should have desired accuracies. One research [74] in online federated NAS used an evolutionary NAS algorithm to reduce the computational cost and communication cost. FedNAS [75] implemented differentiable NAS algorithm DARTS which is explained in Section 3.1.

DecNAS [76] is a population-based offline federated NAS that follows a well-defined process. It begins by initializing a global model and assigning resource budgets (latency or computational cost) to clients. Simplified global models are created by neural network pruning and distributed to client groups. Each communication round executes local training, evaluation of test accuracies on validation datasets, and sending results to the server. The server aggregates local models, calculates weighted accuracies, removes poorly performing models, and updates the remaining models with aggregated gradients. This iterative process repeats until convergence. The global model is then replaced and stored, and this continues until convergence is achieved. Finally, federated training is performed on the stored global model. The method incorporates model pruning and client sampling for communication efficiency but acknowledges challenges in accurately representing test accuracies, particularly in non-IID client data scenarios.

FedNAS [75] is a gradient-based method, inspired by DARTS, that employs a global model known as the supernet, consisting of repeated directed acyclic graphs (DAGs) with all candidate operations. Using relaxation tricks [77], a weighted sum of candidate operations becomes differentiable, facilitating direct updates of architecture parameters via gradient descent. The process involves server initialization, clients downloading and locally updating the supernet, and iterative parameter uploading for weighted averaging by the server until convergence.

RT-FedEvoNAS [74] is a real-time multi-objective evolutionary approach that reduces additional communication costs and computational resource overhead while preserving model performance in the optimization process. The proposed method employs a double-sampling strategy, where submodels are sampled from a global model shared by all individuals within a population, and participating clients are sampled for submodel training. This approach enables the integration of one generation of evolutionary optimization within a single communication round, resulting in a significant reduction in both communication and computation costs.

FedorAS [78] conducts NAS in a federated and resource-aware way, considering heterogeneous devices and data. Clients are clustered based on capabilities, and a supernet is designed as both a search space and a weight-sharing system. The supernet is partially shared with clients, considering their computational and bandwidth capacities. The authors utilize resource-aware one-shot path sampling for lightweight NAS on devices, optimizing neural networks for resource efficiency. The federated training of the supernetwork produces pre-trained networks, reducing the need for on-device training rounds.

A recent study particularly created to support heterogeneous DNN models in distributed edge computing systems developed the ENASFL framework [79], which combines federated learning with neural architecture search. This method utilizes RL to efficiently search and optimize neural architectures across decentralized edge devices, resulting in improved model accuracy and compression rates.

## 4. Network-aware federated neural architecture search (NAFNAS)

In this section, we present a comprehensive design framework for a Federated Neural Network Pruning system featuring an emulated network environment. Our work introduces NAFNAS, an open-source federated learning framework designed for NAS tasks under diverse network conditions as illustrated in Fig. 4. Through model pruning and a novel client grouping algorithm, NAFNAS reduces communication costs and improves convergence time while considering network bandwidth limitations.

### 4.1. System definition

Our goal is to utilize NAS in a distributed environment to search for lightweight DNN models that are able to fit in resource-constrained edge devices. To achieve our goal, we have utilized a neural network pruning algorithm and FL framework to enable federated NAS.

In an environment where the cooperation of FL and NAS is realized, there are servers, that are responsible for managing the search process, and heterogeneous clients that contribute to their local data. A federated NAS solution can be realized by distributing the search process to the clients or by keeping the search on the server and utilizing the clients for the federated training. In this context, our framework takes the second approach executes NAS as a central server, and allows federated fine-tuning to happen on the clients.

We placed a neural network pruning algorithm to the server that shrinks a model iteratively until met with the constraints of the edge devices. An initial model is federated and trained in the distributed environment to generate the pre-trained model. After a pre-trained model is successfully trained, the server conducts pruning iteratively. In each *iteration* of pruning, the server calculates a resource constraint and generates multiple candidate models that fit the constraint. Moreover, the pruning process is controlled by the constraint that is defined by $IRR$ (initial resource reduction) and $RDD$ (resource reduction decay), which are the parameters used for defining the desired resource target in an iteration with referencing to an initial model resource.

Pruned models are fine-tuned in federated settings with the participation of multiple clients. *Number of clients* are defined by practitioners with respect to resource limitations of the experimental setup or the emulation that is desired. Each client has their own private dataset

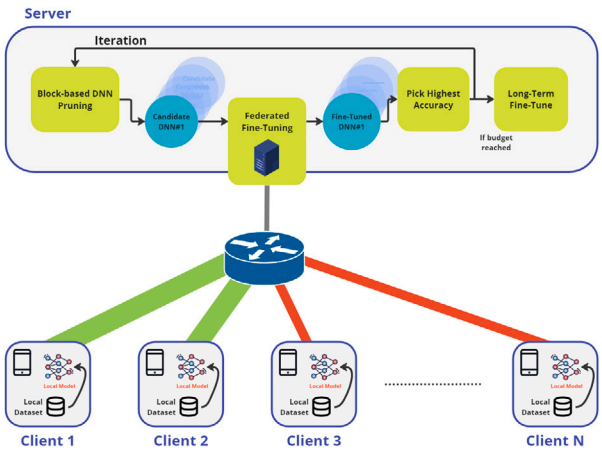



**Fig. 4.** The NAFNAS framework incorporates a federated privacy-preserving flow of actions where clients with heterogeneous network conditions contribute to the model pruning phase occurring at the central server.

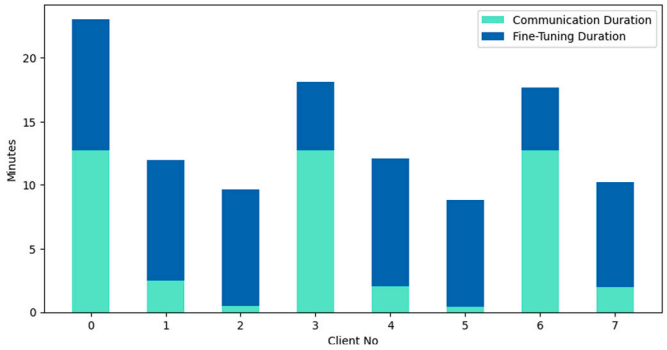



**Fig. 5.** Communication and fine-tuning durations for clients in a group assigned for single model pruned in 1st iteration.

where none of the data samples are shared with the server or other clients. Dirichlet distribution is used for splitting a dataset into non-IID partitions where each client has one of the partitions. The level of non-IID is controlled by the parameter $\alpha$ whose smaller values generate more diverse splits.

Throughout the process, there are many *communication rounds* between the server and clients since the cooperation of the pruning algorithm and federated learning. Those communication rounds are defined by practitioners in order to control the convergence of the global DNN model. Moreover, the clients have heterogeneous resources in terms of computation and network characteristics. The clients with limited *network bandwidth*, in our framework bandwidth, is dynamically defined, and would require more time in the communication while receiving global models and sending updates. NAFNAS includes network emulation for testing and optimizing performance in various scenarios. By studying the impact of network conditions on federated learning, this work offers valuable insights for building resilient and efficient systems.

However, FL cannot be thought of without considering the heterogeneity of clients in terms of the network properties of each. Since there may be low bandwidth clients involved in FL, every process tends to encounter a bottleneck in the communication durations. The Fig. 5 shows that low bandwidth clients such as 0th, 3rd and 6th require more communication time. The client devices have limited computational resources in terms of CPU and memory, therefore the fine-tuning duration would be significantly longer than fine-tuning on a server machine with GPU acceleration.

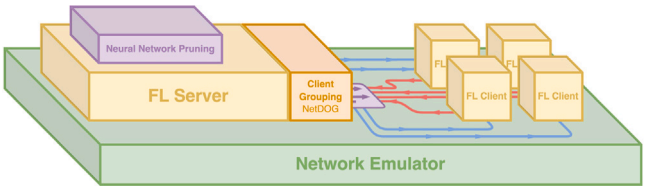



**Fig. 6.** Overall NAFNAS System Architecture.

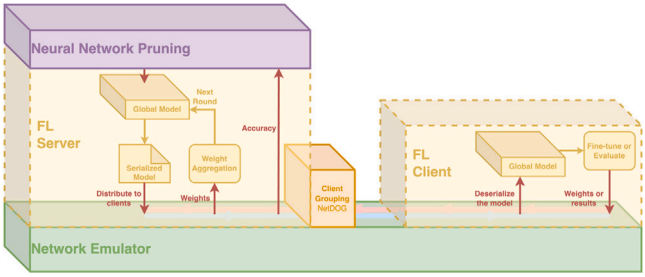



**Fig. 7.** FL server and clients operations in the NAFNAS.

Even in a federated fine-tuning process that needs to be conducted multiple times for more than one DNN, as required by some NAS approaches, the network becomes more crucial. That is why we have implemented a client grouping algorithm, called NetDAG, that groups clients as the number of DNNs needs to be trained/fine-tuned and assigns groups to proper DNNs. To prevent the accuracy drop and optimize the convergence time, the algorithm both considers class label distributions and network bandwidths of clients.

### 4.2. Overview of NAFNAS framework

The designed system consists of the fusion of a neural network pruning algorithm, an FL framework, and a network emulator for comprehensive network simulation. The neural network pruning algorithm generates pruned candidate DNNs and utilizes an FL server to conduct federated fine-tuning. The FL framework is built upon the network emulator in order to enable an enhanced experiment environment in network topologies. The overall experiment system is illustrated in Fig. 6.

The network pruning algorithm generates multiple pruned candidate DNNs by pruning the CNN blocks individually. In each iteration, the best performing DNN from candidates is selected by fine-tuning all candidates in federated settings. For that purpose, each candidate is distributed by a server to enable federated fine-tuning. After DNNs are fine-tuned, their accuracies are collected for selecting the best-performing model.

Before continuing with the network emulator, our implementation of client grouping algorithm NetDAG was placed within the FL server to guide the server to select clients to distribute the DNN models. The algorithm behaves as an intermediate operator that collects bandwidth information from the network and class distributions of clients from the server. Then uses that knowledge to guide the server.

When the FL server is utilized for a candidate DNN, the server serializes the global model (candidate model for pruning algorithm) and distributes the global model to clients. The client either fine-tunes or evaluates the received model and sends updated weights or evaluation results to the server. The network topology is built upon the network emulator with dynamic bandwidth settings. The details of federated learning on FL framework are shown in Fig. 7.

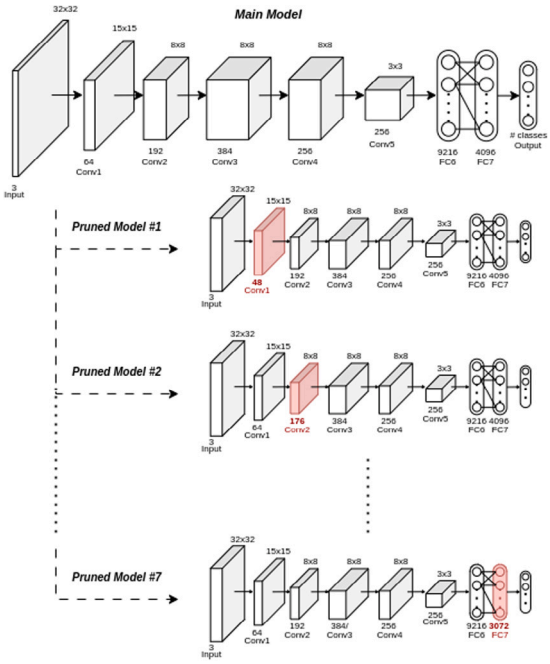



**Fig. 8.** Pruning of AlexNet architecture.

### 4.3. Components of NAFNAS

The system is based on the fundamentals of FL, in terms of decentralized DNN model training with local datasets and its cooperation with neural network pruning algorithms that focus on pruning filters in each layer of CNN iteratively, aimed at reducing the neural network's size and computation cost while maintaining its performance. The federated learning infrastructure is delicately built, comprising server and client components. An emulated network setup is incorporated to simulate federated learning processes on a network, with the benefits of testing and debugging on containers.

#### 4.3.1. Neural network pruning algorithm

The neural network pruning algorithm is the key component to reducing the computational cost in an environment with heterogeneous edge devices. The pruning of the DNN model is done in the server with the contribution of client data.

Our framework is general enough to incorporate various NAS algorithms. However, in this research, we employed NetAdapt [12] due to its iterative mechanism that requires federated fine-tuning multiple times and reputation in the literature. The approach in NetAdapt is structured block-based pruning and each iteration proposes multiple DNN candidates. The original algorithm of NetAdapt uses multiple GPUs for fine-tuning each candidate DNN. Specifically, to finish an iteration and select the best-performing DNN, the algorithm creates subprocesses for each candidate DNN and assigns one available GPU for each of them. Therefore, the process of the NetAdapt algorithm is suitable for utilization in the federated learning system.

The NetAdapt algorithm prunes DNN architecture by its block, the term "block" typically refers to a group of layers or a set of operations that are combined to form a building block of the overall CNN architecture. In our case, we have utilized AlexNet [80] architecture in our pruning process. There are 2 types of blocks in AlexNet architecture, blocks with convolutional layers and blocks with fully connected layers(FC). There are 7 blocks in AlexNet architecture therefore in each iteration NetAdapt algorithm proposes 7 different pruned DNN candidates. A sample for the first iteration is illustrated in Fig. 8.

Along the iterations, the predefined constraint is reduced gradually. An iteration starts with calculating the resource constraint. The resource of the best model of the iteration defines the resource constraint of the next iteration. As described in Section 2, a resource reduction schedule is enabled and there are parameters to be set by a human that define the constraints.

$$\mathcal{R}_{target} = \mathcal{R}_{current} - IRR * (RRD^{i-1}) \tag{3}$$

where $i$ is the number of iterations, $\mathcal{R}_{target}$ and $\mathcal{R}_{current}$ are the target resource of the next iteration and current iteration of the pruned best model respectively. $IRR$ is the initial resource reduction and we have set it as $5 \times 10^{-2}$. $RDD$ is the resource reduction decay that is set as 0.96.

#### 4.3.2. Federated learning framework

Our aim in this research is to accomplish NAS in a privacy-preserving manner. Therefore, FL is an ideal approach to implement in our framework. We utilized federated learning in the fine-tuning step of the pruned model in each iteration. The server executes the pruning algorithm and generates pruned candidate DNNs for fine-tuning, then distributes those DNNs to client devices for fine-tuning with local data. Each pruned candidate DNN requires a fine-tuning process and a federated fine-tuning strategy can be developed. The straightforward approach is sequentially fine-tuning each model in federated settings. The implementation of federated fine-tuning is illustrated in Fig. 4.

To perform federated learning with server and client communication, we implemented Flower FL Framework [81]. Flower is a user-friendly FL framework that enables the creation of servers and clients practically for a federated learning pipeline and also provides scalable and secure communication with gRPC protocol between servers and clients.

The Flower FL framework provides FL strategies, that dictate how the learning process happens, and built-in weight aggregation algorithms to implement and utilize on the server side. Moreover, it is possible in the framework to customize the federated learning pipeline. The transferred data in the communication can be customized and useful information about the federated learning process or the algorithm can be shared with the clients. Also, the behavior of the clients in training and evaluation steps can be customized due to the requirements of the whole experiment.

In the Flower framework, custom configurations have been implemented to enhance communication during the federated learning process and adapt federated learning in the neural network pruning scenario.

Traditionally, servers typically send only global model weights to the clients for training and updating. However, our iterative neural network pruning algorithm generates different DNN architectures. Therefore, the server has been configured to transmit not only the model weights but also the entire model architecture. This is achieved by serializing the PyTorch model using TorchScript. By serializing the model architecture, the server is now capable of sending a comprehensive representation of the neural network structure to the clients, enabling a more detailed exchange of information.

On the other hand, clients have also been configured to deserialize the received architecture information in bytes. To this end, since the federated learning is conducted in several rounds, the server only sends the weights and architecture together in the first round to inform the clients about the architecture they are working on.

This approach can also be advantageous in various scenarios where the server requires access to the complete model architecture for tasks such as dynamic architecture-specific changes during the federated learning communication rounds.

**Federated Metric Aggregation.** The aggregation of performance metrics, such as accuracy, combines the locally computed accuracy values from clients to obtain a global or federated accuracy measure. This process is crucial for evaluating the overall performance of the

global model across the clients where there is not any shared validation data. The federated accuracy is defined where $n_i$ is the total number of training samples if client $i$, as;

$$\text{Aggregated Accuracy} = \frac{\sum_{i=1}^{N} n_i \cdot Acc_i}{\sum_{i=1}^{N} n_i}. \quad (4)$$

Privacy protection is achieved by keeping data on client devices and only sharing model updates. The FL framework uses secure methods like encryption and secure multi-party computation to keep these updates confidential. Also, it is possible to integrate differential privacy by adding noise to the updates, protecting sensitive information. Additionally, Flower uses encrypted channels for secure communication. These features make Flower a secure and privacy-focused federated learning framework that makes the framework beneficial for our NAS framework, ensuring client data stays protected and compliant with privacy regulations.

#### 4.3.3. Network emulator

The cooperation of FL and NAS is feasible for ensuring computational efficiency in a privacy-preserving environment. However, there is a gap in the literature which is the effect and efficiency of communication duration in such an iterative FL process. To observe the behavior of the heterogeneous network capacity of clients, it is needed to implement a sophisticated network emulator.

One of the distinctive approaches of our research is a highly capable and realistic network emulator implementation to federated neural architecture pruning. Federated learning cannot be thought of without network conditions and architectures. A network emulator simulates real-world network conditions, including varying bandwidth, latency, and packet loss. This ensures that the federated learning system performance is tested under conditions that the actual deployment environment. Federated learning usually involves resource-constrained devices. Network emulators enable testing management of resources such as bandwidth and computational power by the system in different network conditions.

We employed CORE, the Common Open Research Emulator [82], which is developed by Boeing Research and Technology, and stands out as a powerful network emulation tool for research, development, and education. As an emulator, different from a simulation where abstract models are used, CORE sets up a representation of a real network that behaves in real-time.

The CORE emulator has demonstrated robust performance in various studies [83,84]. Our server's CPU is significantly more powerful than the one used in the original tests [82] where CORE handled up to 300,000 packets-per-second with up to 120 nodes, constrained primarily by packet operations per second rather than packet size or number of hops, highlighting efficient kernel-level handling. Additionally, CORE has been effectively used in real-time applications [85–87], further validating its reliability and applicability for real-time network emulation.

By using the CORE framework, we have selectively employed various network topology components to create targeted and realistic scenarios. Particularly, we have focused on utilizing custom nodes and switches as the primary components within our federated learning network configurations. Custom nodes afford us the task-specific network devices for our research requirements. Moreover, we have integrated Docker containers within these custom nodes, leveraging containerization for the deployment of flower servers and clients. This approach allows us to develop a seamless incorporation of the containerized Flower FL framework within the broader network topology.

In our experimental setup, we have incorporated the Flower FL framework, specifically utilizing servers and clients encapsulated within Docker containers. For fine-tuning DNNs on the client side, we have imposed strict resource constraints on the client containers by allocating 8 CPU cores and 6 GB of memory. This restriction aims to simulate realistic computing conditions and assess the pruned DNN performance within resource-constrained environments. Our network configuration within CORE encompasses a total of 56 to 210 clients. Each client does not contribute to the distributed fine-tuning process every time, the client selection and grouping are discussed in Section 5.

We have defined multiple tiers of bandwidth conditions and each tier is assigned to each link between the clients and a switch that enables server and clients to connect. A dynamic adjustment mechanism is introduced to the network configurations wherein the bandwidth for each link in the network is modified every 5 min within the predefined limits of the assigned tier. We purposely adjusted the bandwidth levels to imitate real-world scenarios where bandwidth fluctuation occurs. This variation allows us to assess how effectively the federated learning model behaves in different bandwidth situations.

### 4.4. Proposed client grouping algorithm: NetDAG

The proposed client grouping algorithm, named NetDAG (**Net**work and **D**istribution **A**ware Client **G**rouping), is designed to enhance the efficiency of model training in federated learning settings by considering both network bandwidth conditions and non-IID class distribution of clients. The objective behind this algorithm is strategically assigning clients to different DNN models (in the context of neural network pruning) based on network bandwidth and the class distributions of respective datasets.

The algorithm optimizes communication efficiency by assigning clients with lower network bandwidth to models with smaller sizes. Especially assigning to a group that desired to train a specific model. This strategic assignment aims to alleviate potential communication bottlenecks by shifting the communication overhead to smaller models, in order to reduce communication time and optimize the overall federated learning process.

Furthermore, the NetDAG takes into account the non-IID datasets of clients with varying class distributions. To address this challenge, the algorithm employs an iterative process that optimizes the class distribution within each client group. This ensures that each global model that is trained federally receives a diverse and balanced set of data representations, providing a robust and generalized global model. The NetDAG iteratively transfers clients between groups based on the EMD values of the groups, aiming to achieve a balanced and representative distribution of classes within each group and reduce the average EMD value of groups.

The pseudocode of NetDAG is given in Algorithm 1. The algorithm initiates by assigning clients to groups using a simple network-aware approach. It then iteratively refines these assignments over a specified number of iterations (*MAX_ITER*) to achieve enhanced optimization. Within each iteration, the algorithm cycles through a sequence of groups that is changed in each iteration to give priority to each group, exploring potential client transfers between neighboring groups. It strategically evaluates these transfers based on their impact on the EMD value of the groups after the transfer. The algorithm employs a coefficient (*COEFF*) to regulate the transfer. The algorithm expects an improvement as the coefficient in average EMD value if the clients in a transfer pair are in different bandwidth tiers, in other cases expectation is just an improvement in average EMD. That mechanism provides a significant improvement in EMD while also considering bandwidth constraints. This iterative process, guided by EMD and network awareness, ultimately aims to produce client groupings that exhibit a balance between network efficiency and class distribution uniformity.

The Algorithm 1 has individual clients ($c_i$) assigned to specific groups ($g_i$) with bandwidth tiers ($bw_i$). It holds on the entire set of clients ($C$) and its subsets ($C_j$) based on group number $j$. Similarly, it considers the complete set of groups ($G$) and group subsets ($G_j$). Class distributions are captured for individual clients ($Dist_{c_i}$), their assigned groups ($Dist_{g_i}$), the entire dataset ($Dist_G$), and group subsets ($Dist_{G_j}$). Temporary variables ($g_{temp}$ and $G_{temp}$) help in exploring client transfers. Algorithm behavior is controlled by the maximum iteration count ($MAX\_ITER$), transfer threshold coefficient ($COEFF$), total number of groups ($N$), and window length ($w$) used in calculations.

**Algorithm 1** NetDAG: Network and Class Distribution Optimized Client Grouping Algorithm

**Input:** *MAX_ITER*: Maximum number of iterations. $N$: Number of groups. *COEFF*: Coefficient for client transfer threshold.
**Output:** $G$: Group numbers of each client.

$G \leftarrow$ Assign by simple network-aware grouping
$sequence\_of\_groups \leftarrow [1, 2, 3, ..., N]$
**for** iter = 1 **to** *MAX_ITER* **do**
  $sequence\_of\_groups \leftarrow$ shift the sequence
  $noOperationsDone \leftarrow 0$
  **for** $K$ **in** $sequence\_of\_groups$ **do**
    $EMD_\mu \leftarrow \frac{1}{N}\sum_{n=1}^{N} \text{EMD}(Dist_{G_n}, Dist_G)$
    $neighbourClients \leftarrow \{C_{K-w}, ..., C_{K+w}\}$
    $operations = \{\}$
    **for** $c_i$ **in** $C_K$ **do**
      **for** $c_j$ **in** $neighbourClients$ **do**
        $G^{temp} \leftarrow transfer(c_i, c_j)$
        **if** $bw_i = bw_j$ **then** coeff = 1
        **else** coeff = COEFF
        **if** $EMD(G_{g_i}^{temp}) < EMD(G_{g_i}) \times coeff$ **then**
          $EMD'_\mu \leftarrow \frac{1}{N}\sum_{n=1}^{N} \text{EMD}(Dist_{G_n^{temp}}, Dist_G)$
          operations.insert(($c_i$,$c_j$,$EMD'_\mu$))
    $c_i$, $c_j$, $EMD' \leftarrow min(EMD'_\mu)$
    **if** $EMD' > EMD_\mu$ **then**
      $G \leftarrow transfer(c_i, c_j)$
      $noOperationsDone++$
  **if** $noOperationsDone = 0$ **then** break

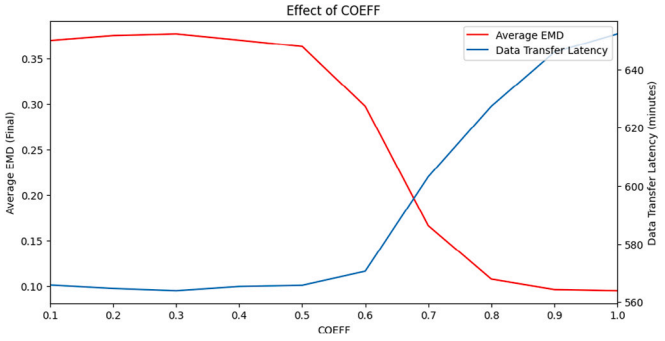
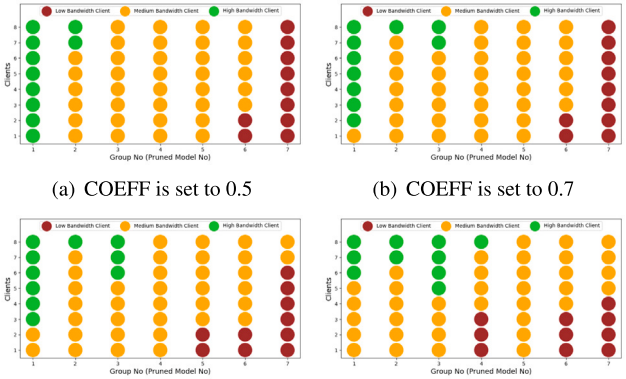



**Fig. 9.** Effect of COEFF on EMD and Data Transfer Latency ($\alpha = 0.3$).

#### 4.4.1. Analysis of NetDAG

There are a couple of parameters that have a direct effect on the outcomes of the NetDAG. $\alpha$ parameter controls the distribution of classes and the algorithm aims to balance the distribution inside the group. Selection of the *COEFF* parameter is crucial for allowing transfers to happen between groups. Therefore, it prohibits the possible outcomes of the algorithm in terms of the groups and their distributions. Also *number of clients* that are involved in the FL process is important since it affects the number of operations that are going to be evaluated.

Our experiments investigated the impact of the *COEFF* on average EMD value and data transfer latency. The computation of the client grouping is conducted on the server, therefore the overhead occurs on the server. Our central server has an Intel Xeon CPU with 2.30 GHz 64 cores and 128 GB of memory. As depicted in Fig. 9, increasing the *COEFF* value resulted in a corresponding decrease in the average EMD value and a concurrent increase in data transfer latency. This phenomenon can be attributed to the relaxed constraint imposed by a higher *COEFF* value. While it allows for the transfer of clients with disparate bandwidth tiers, this flexibility fosters the emergence of bottlenecks within individual groups. However, this relaxed constraint also facilitates the achievement of lower average EMD values. Consequently, the findings clearly show a trade-off between achieving minimal average EMD and maintaining low data transfer latency. Also in Fig. 10, you can find the placements of clients with 3 types of bandwidth tiers in each group for different *COEFF* values.




(a) COEFF is set to 0.5 (b) COEFF is set to 0.7

(c) COEFF is set to 0.9 (d) COEFF is set to 1.0

**Fig. 10.** Groups generated by NetDAG in different COEFF settings.

To show the effect of COEFF on bandwidth variety in groups, we have visualized them by changing *COEFF* in [0.5, 0.7, 0.9, 1.0]. Fig. 10 illustrates that investigation. As *COEFF* increases, groups exhibit more and more bandwidth heterogeneity, which causes bandwidth bottlenecks in more groups.

## 5. Experiments and results

### 5.1. Experiment settings

**Dataset.** CIFAR10 is a widely used dataset in the world of computer vision and machine learning. It contains 60,000 color RGB images, each of them sized 32 × 32 pixels, in 10 classes such as airplanes, cars, birds, and cats. The dataset's relatively low image resolution and diverse content make it a challenging but valuable resource for developing accurate and adaptable models. CNNs are often utilized to handle the complexities of this dataset, establishing CIFAR-10 as a key benchmark for evaluating and comparing different DL approaches in image recognition.

**Model.** AlexNet [80] is a deep CNN architecture designed for image classification tasks. The model consists of eight layers, including five CONV layers and three fully connected layers. AlexNet was designed and trained on the ImageNet dataset, with samples of 3 × 256 × 256 input size. However, we modified the AlexNet for the Cifar10 dataset whose samples are sized in 3 × 32 × 32 by reducing the stride and filter sizes.

**Clients and Resource Type.** We have extended our experiments with different numbers of clients to test the scalability of the framework. Therefore we conducted the experiments with 56 and 210 clients. For the sake of simplicity, most of the illustrations are drawn for 56 clients. *MAC* (Multiply-Accumulate Operations) is a computational operation that accounts for multiplication and addition in a single step. In CNNs, MAC operations happen in the CONV layers where kernels are applied to input feature maps. Therefore, MAC operations are a critical component for measuring the computational efficiency and effectiveness of deep learning models.

**Hardware Settings.** We have utilized computational resource-limited containers as edge devices or clients in our experiments. Each client has 8xCPU Cores and 6 GB of memory, which is a very limited constraint for a case such as CNN training. All the experiments were conducted on the HPC server of Boğaziçi University.

**Table 1**
Bandwidth tiers with limits and step sizes in MBps.

| Minimum (MBps) | Maximum (MBps) | Step Size (MBps) |
|---|---|---|
| 2.5 | 3.5 | 0.5 |
| 4.0 | 5.5 | 0.5 |
| 6.0 | 8.0 | 0.5 |
| 8.5 | 12.5 | 1.0 |
| 13.0 | 16.0 | 1.0 |
| 17.0 | 25.0 | 1.0 |
| 26.0 | 30.0 | 1.0 |
| 31.0 | 34.0 | 1.0 |
| 35.0 | 40.0 | 1.0 |
| 41.0 | 55.0 | 1.0 |
| 46.0 | 65.0 | 1.0 |
| 66.0 | 75.0 | 1.0 |
| 75.0 | 100.0 | 1.0 |
| 100.0 | 400.0 | 20.0 |

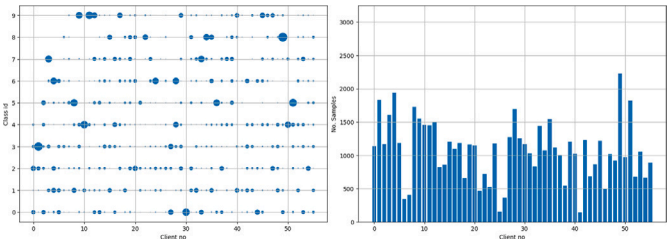


**Fig. 11.** Class distributions and the total number of samples of each client when $\alpha = 0.3$.

**Network Configuration.** To emulate the near real-world scenarios, we have implemented dynamic bandwidth change to the federated learning environment with the help of the network emulator. There are 14 highly heterogeneous tiers of bandwidth levels in our experiments, as given in Table 1, and each tier has an upper and lower limit and step that defines the array of bandwidth values between limits. With the 5-minute update frequency, the bandwidth of every client is randomly selected from an array. For some of the illustrations, we have conducted experiments with 3 bandwidth levels; Low (2.5–3.5 Mbps), Medium (8–32 MBps), and High (50–100 MBps).

**Baseline.** In order to reveal the federated neural network pruning capability of NAFNAS and to compare the improvements achieved using NetDAG in terms of accuracy and bandwidth usage, we have selected two well-known federated learning (FL) algorithms: FedAvg and FedProx. FedAvg was chosen for its foundational role in federated learning, providing a baseline for understanding the impact of pruning and NetDAG on model aggregation and performance. FedProx, on the other hand, was selected for its ability to handle heterogeneous data and client capabilities, allowing us to assess the effectiveness of NetDAG under more challenging real-world conditions. By using these algorithms, we aim to evaluate how NetDAG enhances both model accuracy and communication efficiency across different FL scenarios.

### 5.2. Non-IID dataset generation of clients

We have aimed to split the Cifar10 dataset to clients in non-IID settings to simulate real-world scenarios in federated learning. Dirichlet distribution is utilized to split the uniformly distributed dataset and assign it to clients in non-IID splits. The level of the non-IID split is controlled by a parameter in the Dirichlet algorithm called $\alpha$. The controller parameters $\alpha$ are used to be selected between 0 and 100 in order to change the level of non-IID.

We have set $\alpha = 0.3$ since it generates highly non-IID datasets for clients and even though the number of samples is more balanced than lower values, clients still vary in terms of the sample sizes. Class distributions of each client and the total number of samples where $\alpha$ is set to 0.3 is given in Fig. 11.

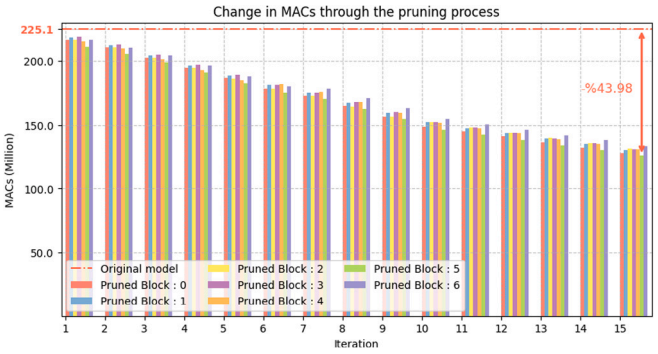



**Fig. 12.** Reduction in MACs.

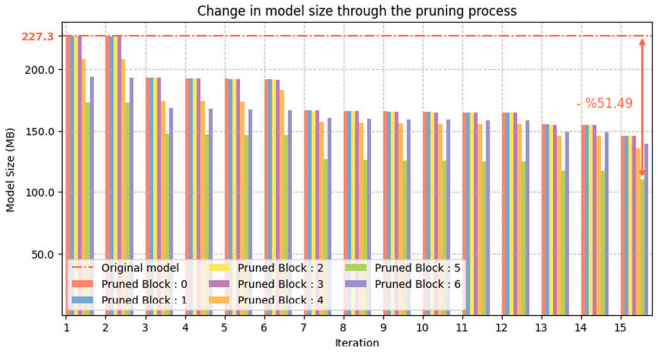



**Fig. 13.** Reduction in model size (MBytes).

### 5.3. Experiments of federated neural network pruning

In the federated neural network pruning process, we have utilized the federally trained AlexNet model as the pre-trained model. The neural network pruning algorithm starts pruning by each layer and selects the pruned models whose resource satisfies the constraint. The resource type is selected as MAC (Multiply-ACCumulate operations) to reduce. The $\mathcal{IRR}$ and $\mathcal{RRD}$ parameters were set as 0.025 and 0.96, respectively and the pruning process continued for 15 iterations.

The federated fine-tuning process in the pruning algorithm is conducted with 56 clients but again the clients are grouped into 7 groups where each group is assigned to one pruned model for federated fine-tuning. The groups are randomly created, therefore any network bandwidth and class distribution aspects are not considered. Each client fine-tuned the assigned model for 50 epochs for 20 rounds.

Through 15 iterations in the neural network pruning, the resource (MAC) is reduced linearly where in each iteration the resource is reduced by an average of 4.27 percent. As shown in Fig. 12, the overall reduction in the resource is %42.98 at the end of the 15nd iteration. Especially, the resource reduction of the 0th and 2nd blocks are higher in each iteration however the selection is not based on the lowest resource, the selection of the best performing model is the pruned model that satisfies the resource constraint with highest accuracy.

On the other hand, the number of model parameters and the model size in terms of megabytes behave differently and reduce more radically. When Figs. 13 is investigated, the total decrease in model size is %51.49 where it is greater than the reduction in MACs. This is because the neural network pruning algorithm decreases the number of filters in convolutional layers or the number of neurons in fully connected layers. The model size highly depends on the number of trainable parameters while the MAC is based on the number of operations.

As shown in Fig. 13, when the last two layers, that are fully connected layers, are pruned, the model size reduces significantly, and if the best model is the model whose 5th block is pruned, the model

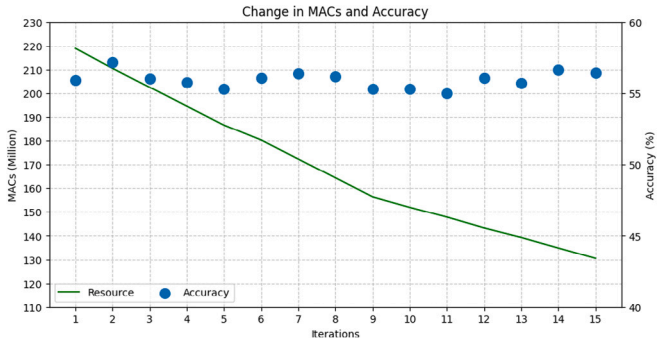



**Fig. 14.** MAC and aggregated accuracies of selected models in each iteration.

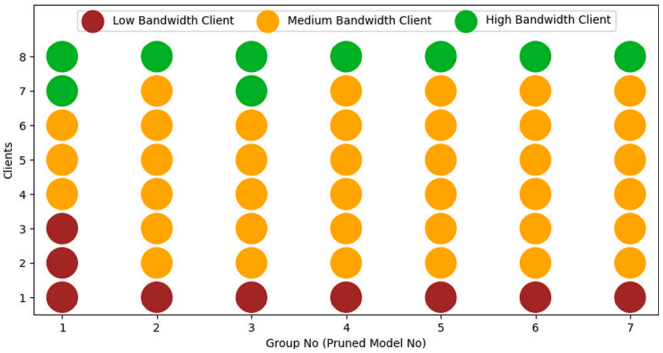



**Fig. 15.** Groups generated by basic grouping.

size drops and therefore the resource constraint in the next iteration drops significantly as well (6rd to 7th iteration in Fig. 13).

The reduction in the model size is important from the communication cost perspective since we have utilized federated learning in an iterative process. Fig. 14 summarizes the experiments by comparing MACs and accuracies of the best-performing models of each iteration. The accuracy is preserved nearly as the model is not pruned at all, however, the reason is that the AlexNet model is capable of easily handling the Cifar10 dataset by the parameter size and the complexity of the model.

### 5.4. Experiments of network-aware federated learning

The strategic assignment of clients with low bandwidth to model groups holds significant importance. Low bandwidth clients, characterized by limited data transfer speed, become bottlenecks in the efficiency of the communication process. In the context of client selection, 2 basic approaches are going to be introduced.

In the *basic grouping*, clients have been selected and placed in the groups randomly as *FedAvg* and *FedProx* select clients randomly. A worst-case example of groups and placements of clients is illustrated in Fig. 15. In basic grouping, to ensure the statistical significance we have conducted multiple experiments.

With considering the network bandwidths of the clients, clients can be grouped for the efficiency of the communication duration. We call that approach *network-aware grouping*. In the network-aware grouping, basically, clients with low bandwidth tiers are placed in groups of smaller model sizes. Therefore, as illustrated in Fig. 16, only the pruned models with smaller sizes face bandwidth bottleneck problems and other pruned models would be executed in shorter communication duration. The overall duration of federated fine-tuning of pruned models would be reduced as well.

Grouping of clients cannot be conducted without considering the class distributions of clients and built groups. The average EMD value

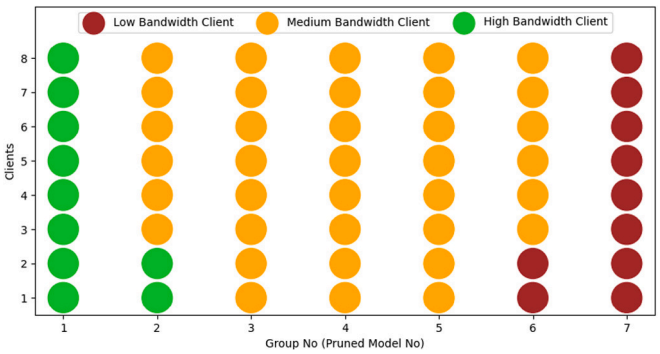



**Fig. 16.** Groups generated by Network-aware grouping.

**Table 2**
Comparison of EMD values by grouping approaches (total 56 clients).

| Group No | FedAvg (Basic Grp.) | FedProx Basic Grp. | Network Aware | NetDAG |
|---|---|---|---|---|
| 1 | 0.58 | 0.41 | 0.58 | 0.32 |
| 2 | 0.57 | 0.38 | 0.61 | 0.17 |
| 3 | 0.62 | 0.69 | 0.65 | 0.21 |
| 4 | 0.49 | 0.60 | 0.63 | 0.23 |
| 5 | 0.42 | 0.49 | 0.32 | 0.33 |
| 6 | 0.41 | 0.45 | 0.61 | 0.21 |
| 7 | 0.67 | 0.69 | 0.50 | 0.26 |
| **Avg** | 0.52 | 0.53 | 0.56 | **0.25** |

increased from 0.52 to 0.56, and most of the EMD values of clients increased from basic grouping to network-aware grouping.

Round duration for pruned models is reduced when the bottleneck is removed. However, the bottleneck still happens for the models with smaller sizes in an iteration. For a sequential federated fine-tuning of pruned models, the overall duration would be reduced significantly. For a parallel federated fine-tuning of pruned models, the duration of a neural network pruning iteration is decided by the pruned model whose federated fine-tuning duration is the longest.

As we move from basic grouping to network-aware grouping, both training and validation accuracies decrease. Although the changes in EMD values between different approaches are relatively small, the drop in accuracy is more pronounced. This decline in accuracy is influenced not only by the non-IID nature of the groups but also by variations in the pruned layers across different grouping methods. Because the change of groups in every iteration results in different pruned models and these models perform differently on validation data, the best-performing model can vary with each iteration. Hence, the performance of the best model in each iteration is likely to be inconsistent.

### 5.5. Network and distribution optimized client grouping

In this subsection, our proposed client grouping algorithm NetDAG has been utilized in the federated neural network pruning. The algorithm parameters *MAX_ITER* and *COEFF* are set as 10 and 0.8 respectively. That means the algorithm can run a maximum of 10 iterations and if the reduction in average EMD value after the transfer is higher than %20 the transfer is permitted. After the EMD values are reduced through the iterations, the transfers between groups become less likely to happen because the reduction possibility decreases. The initial group configuration is set as the simple network-aware grouping in Fig. 16.

The NetDAG has lasted 7 iterations with 56 clients in optimizing network bandwidths and the EMD values in other words non-IIDness of groups. Through the iterations, multiple transfers have occurred and the low bandwidth bottleneck is distributed from 2 groups to 3 groups. That result still holds the low bandwidth clients to groups assigned to the three smallest pruned models. As an example, in Fig. 17, two low

**Table 3**
Comparison of validation accuracy across pruned models.

| No. Clients | Algorithm | 85% AlexNet | 70% AlexNet | 60% AlexNet | 55% AlexNet |
|---|---|---|---|---|---|
| 56 | FedAvg | 56.90 | 55.87 | 54.00 | 53.01 |
| | FedProx | **58.42** | **57.81** | 55.55 | 55.65 |
| | Network Aware | 53.05 | 53.74 | 52.70 | 52.39 |
| | **NetDAG** | 55.77 | 56.21 | **55.70** | **56.46** |
| 210 | FedAvg | **46.39** | 59.86 | 65.65 | 65.92 |
| | FedProx | 31.83 | 38.60 | 39.59 | 38.43 |
| | Network Aware | 43.56 | 56.79 | 65.32 | 65.75 |
| | **NetDAG** | 43.74 | **61.57** | **66.93** | **67.56** |

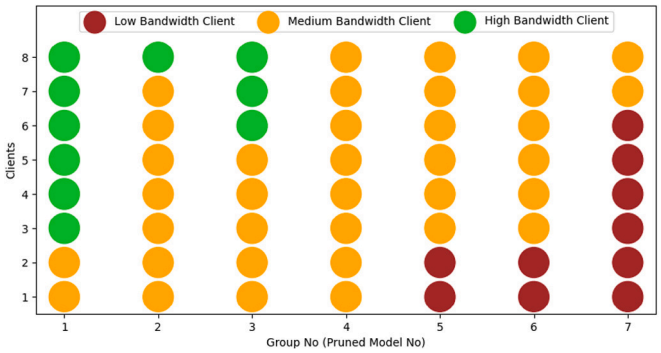



**Fig. 17.** Groups generated by NetDAG.

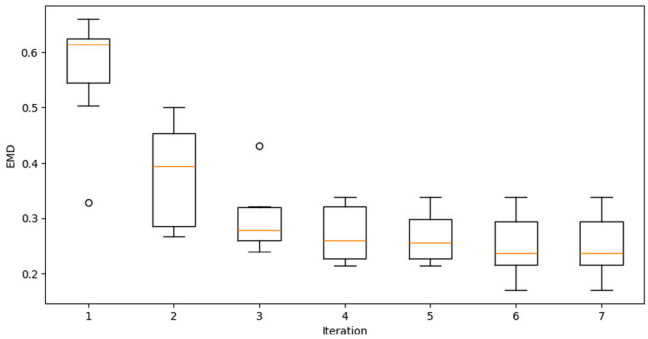



**Fig. 19.** Change in EMD of groups through iterations.

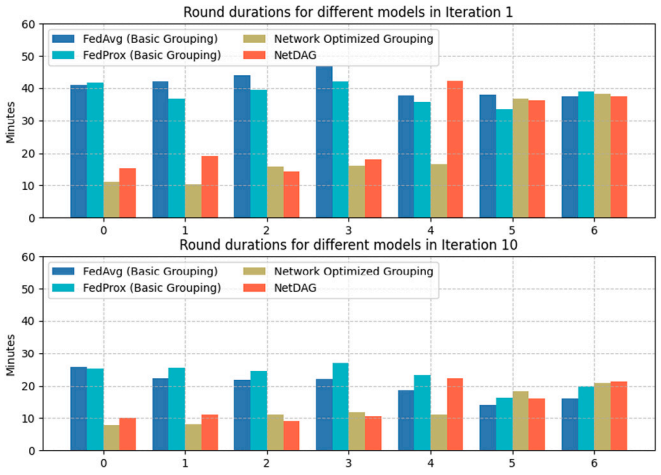



**Fig. 18.** Comparison of overall durations of pruned models across grouping methods for one communication round in 1st and 10nd iterations.

bandwidth clients shifted from other groups. The coefficient was set to 0.8 in that experiment so those transfers were allowed because they reduced the average EMD value by more than 20 percent.

Consequently, the one-round duration of the pruned model has changed, especially one more group sacrificed by placing a low bandwidth bottleneck compared to network-aware grouping. As shown in Fig. 18, pruned models with the numbers 4, 5, and 6 lasted longer since their groups have clients with low bandwidth. When compared with the other grouping approaches, our approach still requires less time than basic grouping, however, since one more group is sacrificed the duration of an iteration increases compared to network-aware grouping.

However, with the help of NetDAG EMD values have been balanced and reduced significantly compared to basic and network-aware grouping. As shown in Fig. 19, EMD values and the average EMD value are significantly reduced through the iterations while sacrificing one group by placing low bandwidth clients in it. The comparison of EMD values of different grouping strategies is given in Table 2. Consequently, our algorithm is effective in reducing the imbalance situation of class distributions in other words level of non-IID.

The experimental results in Table 3 illustrate the validation accuracy of pruned models across different algorithms and varying client numbers. For the scenario with 56 clients, FedProx achieves slightly higher accuracy than NetDAG for the %85 and %70 AlexNet models. However, as the pruning level increases (%60 and %55 AlexNet), NetDAG outperforms both FedAvg and Network Aware, demonstrating its superior adaptability to model size reductions. This indicates that while FedProx is effective with less pruned models, NetDAG provides a balanced approach that maintains high accuracy even as model complexity decreases. The consistent performance of FedProx is because of the set proximal term that ensures better performance in non-IID data distributions.

In the larger scenario with 210 clients, NetDAG consistently achieves the highest validation accuracy across all pruned models. This demonstrates its scalability and effectiveness in handling larger, more heterogeneous networks. FedAvg shows competitive performance, particularly with smaller models, but its accuracy diminishes as the pruning increases. Conversely, FedProx, while strong with fewer clients, struggles to maintain accuracy with 210 clients. These results highlight that NetDAG, as part of the NAFNAS framework, not only enhances model performance but also efficiently manages the heterogeneity of client networks and data distributions. This scalability and robustness make NAFNAS a compelling choice for federated learning environments involving diverse client conditions and significant model pruning.

## 6. Conclusion

In conclusion, this study has introduced a comprehensive exploration of network-aware federated neural architecture search, proposing a novel framework that addresses challenges in federated learning through an iterative neural network pruning algorithm. The developed federated neural network pruning framework demonstrates substantial benefits in handling the challenges of federated DNN pruning. By incorporating network emulation, the framework investigates the impact of network properties on federated learning and the iterative neural network pruning algorithm, successfully ensembling these components.

The key results highlight the crucial role of considering network conditions in federated learning systems. Without accounting for bandwidth, such systems fail to accurately represent real-world scenarios. The study reveals the significant influence of client bandwidth on communication duration during the sharing of global model weight and architecture, contributing to communication overhead and increased convergence time. The developed framework successfully emulates these experimental aspects, providing a distributed neural network pruning framework.

Furthermore, the contribution of this research includes a solution to a key challenge in federated learning settings, the bottleneck created by low bandwidth clients. The proposed algorithm, named NetDAG, introduces intelligent client grouping that considers both non-IID client data and network bandwidth bottlenecks. Utilizing the EMD metric for quantifying non-IID levels, the algorithm optimizes client organization based on class distributions, improving the efficiency and balance of federated learning updates in terms of accuracy and round duration. This holistic framework advances network-aware federated neural architecture search, setting the stage for further investigations into the behavior of other neural architecture search algorithms, the impact of complex non-IID datasets on client grouping, and the exploration of different network condition properties in the context of the emulated federated learning framework.

## CRediT authorship contribution statement

**Göktuğ Öcal:** Writing – original draft, Visualization, Software, Resources, Methodology, Formal analysis, Data curation, Conceptualization. **Atay Özgövde:** Writing – review & editing, Writing – original draft, Resources, Project administration, Methodology, Formal analysis, Conceptualization.

## Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

## Data availability

The data is open-sourced data.

## References


[1] Y. Liu, J. James, J. Kang, D. Niyato, S. Zhang, Privacy-preserving traffic flow prediction: A federated learning approach, IEEE Internet Things J. 7 (8) (2020) 7751–7763.

[2] J. Xu, B.S. Glicksberg, C. Su, P. Walker, J. Bian, F. Wang, Federated learning for healthcare informatics, J. Healthc. Inform. Res. 5 (2021) 1–19.

[3] S. Pandya, G. Srivastava, R. Jhaveri, M.R. Babu, S. Bhattacharya, P.K.R. Maddikunta, S. Mastorakis, M.J. Piran, T.R. Gadekallu, Federated learning for smart cities: A comprehensive survey, Sustain. Energy Technol. Assess. 55 (2023) 102987.

[4] B. McMahan, E. Moore, D. Ramage, S. Hampson, B.A.y. Arcas, Communication-Efficient Learning of Deep Networks from Decentralized Data, in: A. Singh, J. Zhu (Eds.), Proceedings of the 20th International Conference on Artificial Intelligence and Statistics, in: Proceedings of Machine Learning Research, vol. 54, PMLR, 2017, pp. 1273–1282, URL https://proceedings.mlr.press/v54/mcmahan17a.html.

[5] S. Liu, H. Zhang, Y. Jin, A survey on computationally efficient neural architecture search, J. Autom. Intell. 1 (1) (2022) 100002.

[6] B. Zoph, Q.V. Le, Neural architecture search with reinforcement learning, 2016, arXiv preprint arXiv:1611.01578.

[7] J. Tang, D. Sun, S. Liu, J.-L. Gaudiot, Enabling deep learning on IoT devices, Computer 50 (10) (2017) 92–96.

[8] M. Abedi, Y. Iouannou, P. Jamshidi, H. Hemmati, Improving the performance of DNN-based software services using automated layer caching, 2022, arXiv preprint arXiv:2209.08625.

[9] Y. Zhou, S. Ebrahimi, S.Ö. Arık, H. Yu, H. Liu, G. Diamos, Resource-efficient neural architect, 2018, arXiv preprint arXiv:1806.07912.

[10] M. Feurer, K. Eggensperger, S. Falkner, M. Lindauer, F. Hutter, Practical automated machine learning for the automl challenge 2018, in: International Workshop on Automatic Machine Learning At ICML, 2018, pp. 1189–1232.

[11] S. Xu, A. Huang, L. Chen, B. Zhang, Convolutional neural network pruning: A survey, in: 2020 39th Chinese Control Conference, CCC, IEEE, 2020, pp. 7458–7463.

[12] T.-J. Yang, A. Howard, B. Chen, X. Zhang, A. Go, M. Sandler, V. Sze, H. Adam, Netadapt: Platform-aware neural network adaptation for mobile applications, in: Proceedings of the European Conference on Computer Vision, ECCV, 2018, pp. 285–300.

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

[14] C. Li, X. Lin, Y. Mao, W. Lin, Q. Qi, X. Ding, Y. Huang, D. Liang, Y. Yu, Domain generalization on medical imaging classification using episodic training with task augmentation, Comput. Biol. Med. 141 (2022) 105144.

[15] Y. Zhao, M. Li, L. Lai, N. Suda, D. Civin, V. Chandra, Federated learning with non-iid data, 2018, arXiv preprint arXiv:1806.00582.

[16] S.P. Sturluson, S. Trew, L. Muñoz-González, M. Grama, J. Passerat-Palmbach, D. Rueckert, A. Alansary, Fedrad: Federated robust adaptive distillation, 2021, arXiv preprint arXiv:2112.01405.

[17] L.-C. Chen, M. Collins, Y. Zhu, G. Papandreou, B. Zoph, F. Schroff, H. Adam, J. Shlens, Searching for efficient multi-scale architectures for dense image prediction, in: Advances in Neural Information Processing Systems, vol. 31, 2018.

[18] X. Du, T.-Y. Lin, P. Jin, G. Ghiasi, M. Tan, Y. Cui, Q.V. Le, X. Song, Spinenet: Learning scale-permuted backbone for recognition and localization, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2020, pp. 11592–11601.

[19] D. So, Q. Le, C. Liang, The evolved transformer, in: International Conference on Machine Learning, PMLR, 2019, pp. 5877–5886.

[20] B. Zoph, V. Vasudevan, J. Shlens, Q.V. Le, Learning transferable architectures for scalable image recognition, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2018, pp. 8697–8710.

[21] H. Pham, M. Guan, B. Zoph, Q. Le, J. Dean, Efficient neural architecture search via parameters sharing, in: International Conference on Machine Learning, PMLR, 2018, pp. 4095–4104.

[22] P.J. Angeline, G.M. Saunders, J.B. Pollack, An evolutionary algorithm that constructs recurrent neural networks, IEEE Trans. Neural Netw. 5 (1) (1994) 54–65.

[23] K.O. Stanley, R. Miikkulainen, Evolving neural networks through augmenting topologies, Evol. Comput. 10 (2) (2002) 99–127.

[24] C. White, M. Safari, R. Sukthanker, B. Ru, T. Elsken, A. Zela, D. Dey, F. Hutter, Neural architecture search: Insights from 1000 papers, 2023, arXiv preprint arXiv:2301.08727.

[25] E. Real, A. Aggarwal, Y. Huang, Q.V. Le, Regularized evolution for image classifier architecture search, in: Proceedings of the Aaai Conference on Artificial Intelligence, vol. 33, (01) 2019, pp. 4780–4789.

[26] T. Elsken, J.-H. Metzen, F. Hutter, Simple and efficient architecture search for convolutional neural networks, 2017, arXiv preprint arXiv:1711.04528.

[27] T. Elsken, J.H. Metzen, F. Hutter, Efficient multi-objective neural architecture search via lamarckian evolution, 2018, arXiv preprint arXiv:1804.09081.

[28] G. Bender, P.-J. Kindermans, B. Zoph, V. Vasudevan, Q. Le, Understanding and simplifying one-shot architecture search, in: International Conference on Machine Learning, PMLR, 2018, pp. 550–559.

[29] S. Saxena, J. Verbeek, Convolutional neural fabrics, in: Advances in Neural Information Processing Systems, vol. 29, 2016.

[30] H. Cai, C. Gan, T. Wang, Z. Zhang, S. Han, Once-for-all: Train one network and specialize it for efficient deployment, 2020, arXiv preprint arXiv:1908.09791.

[31] H. Liu, K. Simonyan, Y. Yang, Darts: Differentiable architecture search, 2018, arXiv preprint arXiv:1806.09055.

[32] M. Tan, Q. Le, Efficientnet: Rethinking model scaling for convolutional neural networks, in: International Conference on Machine Learning, PMLR, 2019, pp. 6105–6114.

[33] S. Vadera, S. Ameen, Methods for pruning deep neural networks, IEEE Access 10 (2022) 63280–63300.

[34] Z. Zhou, W. Zhou, R. Hong, H. Li, Online filter weakening and pruning for efficient convnets, in: 2018 IEEE International Conference on Multimedia and Expo, ICME, IEEE, 2018, pp. 1–6.

[35] S. Lin, R. Ji, Y. Li, Y. Wu, F. Huang, B. Zhang, Accelerating convolutional networks via global & dynamic filter pruning, in: IJCAI, vol. 2, (7) Stockholm, 2018, p. 8.

[36] L. Zhang, Z. Tan, J. Song, J. Chen, C. Bao, K. Ma, Scan: A scalable neural networks framework towards compact and efficient models, Adv. Neural Inf. Process. Syst. 32 (2019).

[37] C. Gong, Y. Chen, Y. Lu, T. Li, C. Hao, D. Chen, VecQ: Minimal loss DNN model compression with vectorized weight quantization, IEEE Trans. Comput. 70 (5) (2020) 696–710.

[38] M. Lin, R. Ji, Y. Zhang, B. Zhang, Y. Wu, Y. Tian, Channel pruning via automatic structure search, 2020, arXiv preprint arXiv:2001.08565.

[39] H. Li, X. Yue, Z. Wang, Z. Chai, W. Wang, H. Tomiyama, L. Meng, Optimizing the deep neural networks by layer-wise refined pruning and the acceleration on FPGA, Comput. Intell. Neurosci. 2022 (2022).
[40] G.S. Chung, C.S. Won, Filter pruning by image channel reduction in pre-trained convolutional neural networks, Multimedia Tools Appl. 80 (2021) 30817–30826.
[41] Z. Li, H. Li, L. Meng, Model compression for deep neural networks: A survey, Computers 12 (3) (2023) 60.
[42] S. Han, J. Pool, J. Tran, W. Dally, Learning both weights and connections for efficient neural network, in: Advances in Neural Information Processing Systems, vol. 28, 2015.
[43] T.-J. Yang, Y.-H. Chen, V. Sze, Designing energy-efficient convolutional neural networks using energy-aware pruning, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2017, pp. 5687–5695.
[44] W. Jiang, Y. Chen, S. Wen, L. Zheng, H. Jin, PDAS: Improving network pruning based on progressive differentiable architecture search for DNNs, Future Gener. Comput. Syst. 146 (2023) 98–113.
[45] R. Banner, Y. Nahshan, D. Soudry, Post training 4-bit quantization of convolutional networks for rapid-deployment, Adv. Neural Inf. Process. Syst. 32 (2019).
[46] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, D. Kalenichenko, Quantization and training of neural networks for efficient integer-arithmetic-only inference, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2018, pp. 2704–2713.
[47] S.A. Tailor, J. Fernandez-Marques, N.D. Lane, Degree-quant: Quantization-aware training for graph neural networks, 2020, arXiv preprint arXiv:2008.05000.
[48] C. Li, M. Lin, Z. Ding, N. Lin, Y. Zhuang, Y. Huang, X. Ding, L. Cao, Knowledge condensation distillation, in: European Conference on Computer Vision, Springer, 2022, pp. 19–35.
[49] B. McMahan, E. Moore, D. Ramage, S. Hampson, B.A. y Arcas, Communication-efficient learning of deep networks from decentralized data, in: Artificial Intelligence and Statistics, PMLR, 2017, pp. 1273–1282.
[50] A. Hard, K. Rao, R. Mathews, S. Ramaswamy, F. Beaufays, S. Augenstein, H. Eichner, C. Kiddon, D. Ramage, Federated learning for mobile keyboard prediction, 2018, arXiv preprint arXiv:1811.03604.
[51] W. Yang, Y. Zhang, K. Ye, L. Li, C.-Z. Xu, Ffd: A federated learning based method for credit card fraud detection, in: Big Data–BigData 2019: 8th International Congress, Held As Part of the Services Conference Federation, SCF 2019, San Diego, CA, USA, June 25–30, 2019, Proceedings 8, Springer, 2019, pp. 18–32.
[52] N. Rieke, J. Hancox, W. Li, F. Milletari, H.R. Roth, S. Albarqouni, S. Bakas, M.N. Galtier, B.A. Landman, K. Maier-Hein, et al., The future of digital health with federated learning, NPJ Digital Med. 3 (1) (2020) 119.
[53] C. Briggs, Z. Fan, P. Andras, A review of privacy-preserving federated learning for the internet-of-things, Federated Learn. Syst.: Towards Next-Gener. AI (2021) 21–50.
[54] K. Hsieh, A. Phanishayee, O. Mutlu, P. Gibbons, The non-iid data quagmire of decentralized machine learning, in: International Conference on Machine Learning, PMLR, 2020, pp. 4387–4398.
[55] S. Itahara, T. Nishio, Y. Koda, M. Morikura, K. Yamamoto, Distillation-based semi-supervised federated learning for communication-efficient collaborative training with non-iid private data, IEEE Trans. Mob. Comput. 22 (1) (2021) 191–205.
[56] E. Jeong, S. Oh, H. Kim, J. Park, M. Bennis, S.-L. Kim, Communication-efficient on-device machine learning: Federated distillation and augmentation under non-iid private data, 2018, arXiv preprint arXiv:1811.11479.
[57] L. Cai, D. Lin, J. Zhang, S. Yu, Dynamic sample selection for federated learning with heterogeneous data in fog computing, in: ICC 2020-2020 IEEE International Conference on Communications, ICC, IEEE, 2020, pp. 1–6.
[58] Z. Wang, Y. Zhu, D. Wang, Z. Han, Fedacs: Federated skewness analytics in heterogeneous decentralized data environments, in: 2021 IEEE/ACM 29th International Symposium on Quality of Service, IWQOS, IEEE, 2021, pp. 1–10.
[59] Q. Ma, Y. Xu, H. Xu, Z. Jiang, L. Huang, H. Huang, FedSA: A semi-asynchronous federated learning mechanism in heterogeneous edge computing, IEEE J. Sel. Areas Commun. 39 (12) (2021) 3654–3672.
[60] S. Reddi, Z. Charles, M. Zaheer, Z. Garrett, K. Rush, J. Konečnỳ, S. Kumar, H.B. McMahan, Adaptive federated optimization, 2020, arXiv preprint arXiv:2003.00295.
[61] T. Li, A.K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, V. Smith, Federated optimization in heterogeneous networks, Proc. Mach. Learn. Syst. 2 (2020) 429–450.
[62] Y. Teng, Z. Li, L. Yin, N. Wu, State-based differential privacy verification and enforcement for probabilistic automata, Mathematics 11 (8) (2023) 1853.
[63] H.B. McMahan, D. Ramage, K. Talwar, L. Zhang, Learning differentially private recurrent language models, 2017, arXiv preprint arXiv:1710.06963.
[64] S. Ramaswamy, O. Thakkar, R. Mathews, G. Andrew, H.B. McMahan, F. Beaufays, Training production language models without memorizing user data, 2020, arXiv preprint arXiv:2009.10031.
[65] L. Sun, L. Lyu, Federated model distillation with noise-free differential privacy, 2020, arXiv preprint arXiv:2009.05537.
[66] K. Bonawitz, H. Eichner, W. Grieskamp, D. Huba, A. Ingerman, V. Ivanov, C. Kiddon, J. Konečnỳ, S. Mazzocchi, B. McMahan, et al., Towards federated learning at scale: System design, Proc. Mach. Learn. Syst. 1 (2019) 374–388.
[67] Z. Wang, Z. Yang, I. Azimi, A.M. Rahmani, Differential private federated transfer learning for mental health monitoring in everyday settings: A case study on stress detection, 2024, arXiv preprint arXiv:2402.10862.
[68] C. Wang, H. Xia, S. Xu, H. Chi, R. Zhang, C. Hu, FedBnR: Mitigating federated learning non-IID problem by breaking the skewed task and reconstructing representation, Future Gener. Comput. Syst. 153 (2024) 1–11.
[69] D. Li, J. Wang, Fedmd: Heterogenous federated learning via model distillation, 2019, arXiv preprint arXiv:1910.03581.
[70] D. Jiang, C. Shan, Z. Zhang, Federated learning algorithm based on knowledge distillation, in: 2020 International Conference on Artificial Intelligence and Computer Engineering, ICAICE, IEEE, 2020, pp. 163–167.
[71] C. Wu, F. Wu, L. Lyu, Y. Huang, X. Xie, Communication-efficient federated learning via knowledge distillation, Nature Commun. 13 (1) (2022) 2032.
[72] H. Zhu, H. Zhang, Y. Jin, From federated learning to federated neural architecture search: A survey, Complex Intell. Syst. 7 (2021) 639–657.
[73] H. Zhu, Y. Jin, Multi-objective evolutionary federated learning, IEEE Trans. Neural Netw. Learn. Syst. 31 (4) (2020) 1310–1322.
[74] H. Zhu, Y. Jin, Real-time federated evolutionary neural architecture search, IEEE Trans. Evol. Comput. 26 (2) (2021) 364–378.
[75] C. He, E. Mushtaq, J. Ding, S. Avestimehr, Fednas: Federated deep learning via neural architecture search, 2021.
[76] J. Yuan, M. Xu, Y. Zhao, K. Bian, G. Huang, X. Liu, S. Wang, Federated neural architecture search, 2020, arXiv preprint arXiv:2002.06352.
[77] M.A. Trick, A linear relaxation heuristic for the generalized assignment problem, Naval Res. Logist. 39 (2) (1992) 137–151.
[78] L. Dudziak, S. Laskaridis, J. Fernandez-Marques, Fedoras: Federated architecture search under system heterogeneity, 2022, arXiv preprint arXiv:2206.11239.
[79] Z. Zhang, Z. Liu, Y. Zhao, C. Qiu, C. Zhang, X. Wang, ENASFL: A federated neural architecture search scheme for heterogeneous deep models in distributed edge computing systems, IEEE Trans. Netw. Sci. Eng. (2023).
[80] A. Krizhevsky, I. Sutskever, G.E. Hinton, Imagenet classification with deep convolutional neural networks, in: Advances in Neural Information Processing Systems, vol. 25, 2012.
[81] D.J. Beutel, T. Topal, A. Mathur, X. Qiu, J. Fernandez-Marques, Y. Gao, L. Sani, K.H. Li, T. Parcollet, P.P.B. de Gusmão, et al., Flower: A friendly federated learning research framework, 2020, arXiv preprint arXiv:2007.14390.
[82] J. Ahrenholz, C. Danilov, T.R. Henderson, J.H. Kim, CORE: A real-time network emulator, in: MILCOM 2008-2008 IEEE Military Communications Conference, IEEE, 2008, pp. 1–7.
[83] J. Ahrenholz, Comparison of CORE network emulation platforms, in: Milcom 2010 Military Communications Conference, IEEE, 2010, pp. 166–171.
[84] J. Ahrenholz, T. Goff, B. Adamson, Integration of the CORE and EMANE network emulators, in: MILCOM 2011 Military Communications Conference, IEEE, 2011, pp. 1870–1875.
[85] P. Kumar, J. Liu, A.S.M. Tayeen, S. Misra, H. Cao, J. Harikumar, O. Perez, FLNET2023: Realistic network intrusion detection dataset for federated learning, in: IEEE Military Communications Conference, IEEE, 2023, pp. 345–350.
[86] M. Stifter, J. Cordova, J. Kazmi, R. Arghandeh, Real-time simulation and hardware-in-the-loop testbed for distribution synchrophasor applications, Energies 11 (4) (2018).
[87] C. Ogilvie, J. Ospina, C. Konstantinou, T. Vu, M. Stanovich, K. Schoder, M. Steurer, Modeling communication networks in a real-time simulation environment for evaluating controls of shipboard power systems, in: 2020 IEEE CyberPELS, (CyberPELS), IEEE, 2020, pp. 1–7.

**Göktuğ Öcal** received his MS degree from Bogazici University, Istanbul, in 2023 and BS degree from Istanbul Technical University in 2020. He worked as Data Scientist in Faradai between 2021-2022 and Data Scientist in Ford Otosan in 2022-2023. His research interestsinclude Machine Learning, Artificial Intelligence, Internet of Things, MLOps, Cloud Computing.

**Atay Ozgovde** received his BS and MS degrees from Bogazici University, Istanbul, in 1995 and 1998, respectively. He worked as an R&D engineer in NortelNetworks between 1998-2001. He completed his Ph.D. degree in the NETLAB research group Bogazici University in 2009. He worked as a full-time faculty at the Galatasaray University Computer Engineering Department between 2009-2021. Currently, he is an assistant professor in the Computer Engineering Department, Bogazici University. His research interestsinclude Computer Networks, Edge Computing, 5G and Beyond, Internet of Things, Ambient Intelligence, SDN and mobile cloud computing. He is a senior member of the IEEE.