POLITECNICO DI TORINO

Master's Degree in Ingegneria Informatica - Artificial Intelligence and Data Analytics

Master's Degree Thesis

Federated Learning Meets Model Compression for Image Classification on Memory-Constrained Devices

Supervisors

Prof. Alessio SACCO

Prof. Flavio ESPOSITO

Prof. Guido MARCHETTO

Candidate

Alessandro MASCI

Ottobre 2024

Abstract

In a world flooded with electronic devices and sensors, the necessity of making them cooperate increases on a daily basis. At the same time, it is crucial to guarantee the safety and privacy of the exploited data. Furthermore, many of the devices that help us in our daily activities are compact appliances with constrained computational and storage capabilities. Therefore, it is essential to find new solutions that enhance memory efficiency without compromising accuracy. Another main aspect that characterizes the effectiveness of these devices is their speed in performing inference: the vast amount of daily-generated data demands increasingly faster inference times. Thus, looking for new solutions to merge all these necessities is crucial.

One of the most exploited methodologies for enabling different devices to collaboratively enhance the reliability of artificial intelligence models is *Federated Learning (FL)*. Thanks to this approach, it is possible to share only locally computed model weights rather than transmitting personal data across the network, preserving privacy.

A strong yet useful approach for decreasing a model's size while maintaining its accuracy is the implementation of Model Compression Techniques, such as *Weight Pruning*: this technique enables the model to use only a selected subset of its weights to perform inference. During the forward step, the model's zero-masked weights are not considered, resulting in a decreased inference time. In this work, we have studied three different pruning strategies: Global Unstructured Pruning (GUP), Local Unstructured Pruning (LUP), and Local Structured Pruning (LSP). Each of the three pruning techniques has been evaluated with various pruning percentages: 20%, 40%, 60%, and 80%.

We then analyzed how these approaches work in a FL environment with a network of 4, 8, or 12 clients arranged in a Ring-All-Reduce topology (first set of simulations) and in a Consensus-Based topology (second set). The experiments aim to investigate the behavior of several neural networks, i.e., ResNet18, ResNet50, VGG16, MobileNetV2, TinyYoloV2, and LeNet5.

The experiments were run in a customized environment developed using Docker and physical GPUs accessible on Chameleon Cloud, a publicly available testbed. In particular, Docker has been used to create a local network with a tailored number of clients and to configure the network topology on a single device.

The findings of this study highlight considerable differences between the pruning techniques and the possibility of indicating the best configuration settings, e.g., type of neural network, number of clients, and network topology. In the majority of scenarios, both GUP and LUP can effectively reduce model size by up to

40% without significantly losing accuracy. Conversely, LSP consistently results in minimal accuracy, even with a modest pruning percentage.

The findings presented in this study offer valuable insights into the potential development and utilization of neural network models on resource-constrained devices, as well as the potential for a federated environment to mitigate the impact of model pruning. Additionally, there is interest in exploring how various pruning techniques interact with different neural networks and their impact on reducing the volume of data transmitted over the network during federated algorithms.

Table of Contents

Chapter 1

Introduction

Today, the world is characterized by the proliferation of thousands and thousands of technological devices, each manifesting distinct features related to their specific purpose. To be more precise, according to the latest available data, there are approximately 17.08 billion connected devices around the world, and this number is expected to almost double by 2030 [1].

Throughout the past century, there has been an essential evolution in technological devices, resulting in their continuous enhancement in terms of intelligence, speed, and multitasking capabilities. The increasing number of functions performed by these devices has contributed to an exponential growth in their utilization. Smart devices have assumed a fundamental role in various fields, such as commercial, environmental, healthcare, industrial, and smart cities. Their applications range from remote patient monitoring, glucose monitoring and heart rate monitoring to air quality and temperature assessment and thousands of other purposes. A common thread among these various applications is represented by the usage and the generation of large amounts of data: the availability of more data leads to more accurate estimations.

A critical aspect of these systems is their necessity to work with extreme transparency for those exploiting them, ensuring their presence is nearly imperceptible. It is imperative that their influence be reduced as much as possible, and this can be achieved by creating small and lightweight devices.

The necessity to use large amounts of data to perform precise computations and the small dimensions of devices are features that are challenging to merge. The reduced size of these devices impacts various aspects, including computational capabilities, storage capacity, and battery consumption. Furthermore, complexity increases when real-time analysis is required, which means that the device must be able to handle a continuous stream of data while performing its operations. The main goal of this study is to find a solution able to handle all these aspects (i.e. power consumption, storage usage, fast computing, and limited device resources) through the strategic application of various Machine Learning methodologies without losing quality in terms of model accuracy.

The first aspect to analyze is the presence of billions of heterogeneous devices. The growing quantity of sensors and appliances allows their integration within a large ecosystem where every client has the possibility to share their knowledge with the final goal of enhancing the performance of a common model on a specific task. This approach is possible thanks to the implementation of *Federated Learning* (2.1). The most significant advantage of Federated Learning is the ability to preserve privacy even if every client exploits its local dataset to train the neural network model before sharing information with other clients.

Secondly, it is crucial to develop a solution that does not require substantial memory due to the small size of the devices mentioned above. Concerning this aspect, *Model Compression Techniques* are powerful instruments that reduce each client's local Machine Learning model size. Multiple Model Compression Techniques exist, each characterized by its own set of advantages and disadvantages, described in 2.2. Among the various Model Compression Techniques, this study focuses on *Weight Pruning* (2.2.1). This strategy is implemented within the PyTorch prune library and implements the pruning strategy by exploiting a bitmask that indicates which weights must be considered during the forward step and which must not. The Weight Pruning strategies exploited are *Global Unstructured Pruning (GUP)*, *Local Unstructured Pruning (LUP)*, and *Local Structured Pruning (LSP)*. Each pruning strategy has been evaluated with four different pruning percentages: 20%, 40%, 60%, and 80%.

Finally, the analysis of the network topology to which clients are connected is another key topic. This study focuses on the implementation of *Ring-All-Reduce* and *Consensus-Based* topologies, each with a variable number of clients. More details about this aspect can be found in 2.1.2 and 2.1.3.

Docker has been utilized to enable the simulation of a large number of clients on a single physical machine. This approach allows for creating a container for each client, making it feasible to assign a distinct IP address to each node and interconnect them based on the selected network topology. To better understand how the simulated environment has been created, refer to 4.1.

The results obtained (4) highlight how these aspects affect the accuracy and inference time when performing an image classification task using the CIFAR10 dataset. Different neural networks have been tested: LeNet5, TinyYoloV2, MobileNetV2, ResNet18, ResNet50, and VGG16. The evaluation metrics chosen allowed us to understand if a particular configuration (topology, number of clients, and pruning strategy) is able to preserve the model's accuracy while decreasing its complexity, memory footprint, and the time necessary to classify a new input image. Dealing with a network of clients in a Federated Learning environment requires tackling another aspect: bandwidth utilization. For this reason, the results section presents two different graphs (Figure 4.14) that show the amount of MB sent by a single client according to the neural network used in a ring topology.

The results show different behaviours among the three different Weight Pruning approaches. When comparing accuracy values, Global Unstructured Pruning is the technique that performs best in every scenario; Local Unstructured Pruning, on the other hand, performs worse than GUP, mainly when the pruning percentage is higher than 40%. Local Structured Pruning, instead, is the approach with the worst behaviour in every simulation set, regardless of the network topology or the number of clients. Conversely, the inference time's values are not particularly affected by Weight Pruning: in most scenarios, the value decreases, as expected, but not enough to justify a decrease in the model accuracy. An interesting aspect that emerged from this study concerns the model's memory usage: by exploiting PyTorch's sparse tensors, it is possible to decrease the file size where the model is saved. In this way, the pruning approach reduces memory consumption, enabling the development of classification tasks on memory-constrained devices.

In conclusion, this study demonstrates the feasibility of optimizing Federated Learning for resource-constrained devices through Weight Pruning techniques with different network topologies. The results show that Global Unstructured Pruning consistently provides the best balance between model accuracy and complexity reduction. However, inference time improvements remain marginal, suggesting that further enhancements are needed to make this trade-off more favourable.

Chapter 2

Background

2.1 Distributed Learning and Federated Learning

The number of smart devices that characterize our lives increases daily, leading to the necessity of upgrading well-known Machine Learning strategies.

A traditional training model approach relies on a centralized server or a local machine where it is possible to use local data and local resources to perform the desired task. This strategy, called *Centralized Learning*, presents various drawbacks; first, data are collected and stored in a single location, raising privacy issues and increasing the data transfer costs to send large data volumes from different devices to the central node. Secondly, the daily growing amount of data generated by sensors and devices makes data storing and processing very challenging: large datasets require more storage and computational resources, leading to higher infrastructure costs and difficulty in scaling up the system.

In response to these challenges, a new approach has begun to be employed: *Distributed Learning* [2]. In Distributed Learning, a large dataset is typically split and shared between multiple machines or nodes that collaborate to train a single model and achieve a common objective. This strategy addresses the scalability issues of Centralized Learning since every client works with a subset of the original dataset: increasing the number of clients within the distributed environment makes it possible to split the original dataset into a larger number of chunks, each with a reduced dimension. However, Distributed Learning is characterized by privacy concerns. Indeed, the necessity of sharing the training set among clients using the network could lead to privacy leaks. Furthermore, Distributed Learning requires frequent and intensive communication between nodes to ensure consistency during the training phase.

An initial solution to mitigate privacy concerns associated with Distributed Learning is represented by *Federated Split Learning* [3]. This algorithm splits the deep learning model into two chunks: front-end and back-end. The front-end segment is located on an edge device that trains it up to a specified point, after which the output is transmitted to a central server that exploits the back-end segment to complete the training phase and generate the output. This solution combines the parallelism of Federated Learning with reduced local memory requirements by splitting the model into segments. The benefits of Federated Split Learning are various, such as lower computation and memory demand for clients, as well as improved scalability since the edge server and the client can train independently. However, a significant challenge of this approach is represented by the strategy chosen to split the model and the selection of the cutting-point, as these factors significantly influence model performance. Additionally, since clients share the model's hidden variables with the server over the Internet, Federated Split Learning is prone to privacy concerns.

This research focuses on a methodology aimed at enhancing privacy: *Federated Learning* [4]. With Federated Learning, each client trains the local model, exploiting its own training set. Once the training phase is completed, every client sends only its model weights to the server. The server has the primary purpose of indicating the neural network model to use for every client, to average the received weights, and to send back to the clients the resulting weights. In this way, the security is strongly enhanced since the raw data never leaves the client's devices. However, every client must store all the layers of its model locally. The memory consumption aspect of this approach is discussed in more detail in the subsequent section of this work (2.2). Furthermore, communication between the server and clients occurs less frequently with respect to Distributed Learning, reducing communication overhead and making Federated Learning suitable for environments with limited bandwidth.

Federated Learning implements three distinct paradigms for training large-scale Machine Learning models: *Parameter Server (PS)*, *Ring-All-Reduce (RAR)* and *Consensus-Based (CB)*. These three strategies are distinguished by different network topologies and the presence or absence of a central server for weight aggregation [5].

2.1.1 Parameter Server

The topology exploited by this strategy is represented in Figure 2.1. This is the most straightforward architecture. In the beginning, each client trains its local model with its local dataset: this fundamental step is present in all strategies and ensures privacy and data accessibility. It is important to note that each client uses the same model, which is communicated to it by the server before starting the training phase. Subsequently, each client transmits the computed weights to the Parameter Server, which then aggregates them to create a new set of weights. At the end of the aggregation step, each client receives the new set of weights from the server and updates its local model. In this way, it is possible to enhance the system's security, as clients only transmit their model weights over the network rather than raw data. On the other hand, the Parameter Server represents a single point of failure for the architecture, and every client has to send the entire set of model weights over the network, increasing the bandwidth utilization.

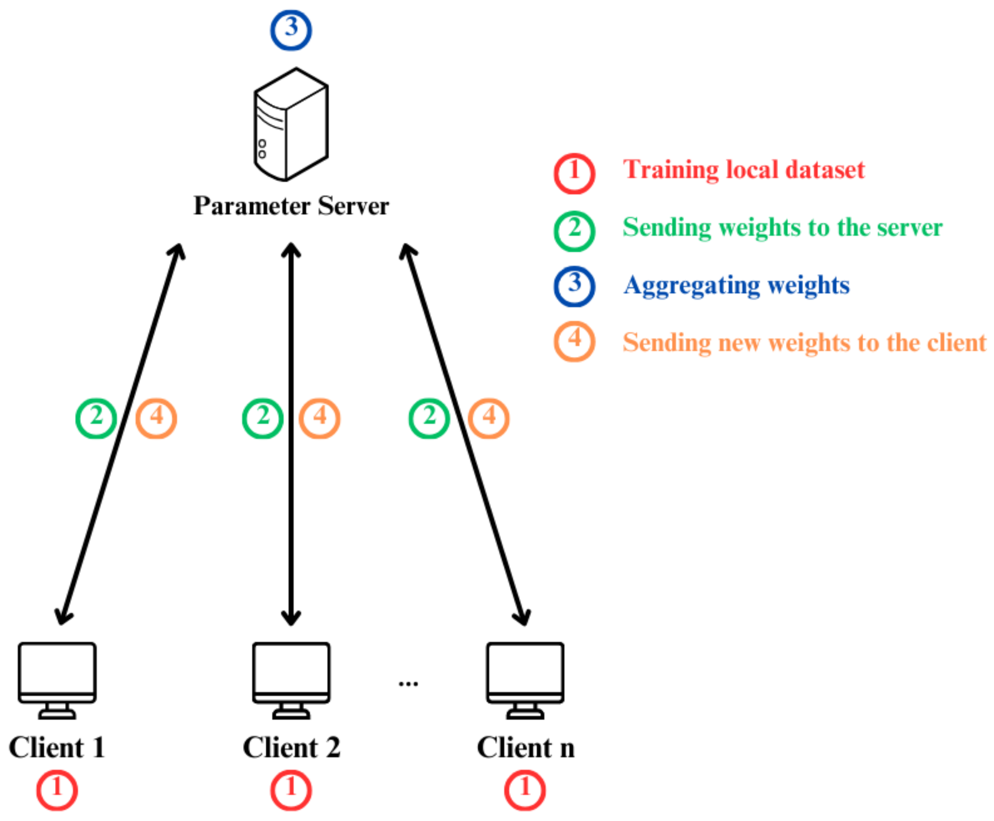


Figure 2.1: Federated Learning process with Parameter Server and n clients

2.1.2 Ring-All-Reduce

The Ring-All-Reduce algorithm consists of two different phases: the share-reduce phase and the share phase. The algorithm is illustrated in Figure 2.2. The first step is the training phase, during which each client trains its model and divides the model weights into chunks equal to the number of clients in the federated environment. Figure 2.2 shows the RAR algorithm with four clients, and, for this reason, every client divides the weights into four segments and assigns a unique ID to each segment. Then, the preliminary step of the share-reduce phase begins, with each client sending the segment associated with the client ID to its successor within the ring topology. After this initial step, each client sends a chunk obtained by reducing the chunk received in the previous step and its own chunk with the same ID to its successor. For example, client 1 received the chunk with ID 4 in step 2; in step 3, client 1 sends a reduced chunk obtained from the chunk with ID 4 received from client 4 (green) and its own chunk with ID 4 (red). In the following step, client 1 sends to its successor a chunk obtained by reducing the chunks with ID 3 (green and blue) received in step 4 and its own chunk with ID 3 (red). During the last step of the share-reduce phase (step 5), every client sends a chunk obtained by reducing all the chunks with the same ID. Now, the share phase starts: each reduced segment (R1, R2, R3, and R4 in Figure 2.2) is shared across the network. At the end of the algorithm (final step), each client is able to update its model weights using four chunks created from the weights of all the clients in the network.

With the Ring-All-Reduce algorithm, bandwidth utilization can be decreased as each client sends and receives the model one chunk at a time. With a larger number of clients, the size of every chunk is smaller. Another fundamental improvement is the absence of a central server during algorithm execution; in this way, a single point of failure does not characterize the network. However, potential bottlenecks can arise within the ring topology, as the algorithm progression is contingent upon the speed of the slowest client in each step, necessitating all clients to wait accordingly.

2.1.3 Consensus-Based

As for the Ring-All-Reduce paradigm, the Consensus-Based paradigm relies on a central server only during the initial steps to indicate the model to use to each client. An example of a Consensus-Based topology with four clients is shown in Figure 2.3. After training the local model, every client sends its model weights to every successor and receives the weights from all the predecessors. The number of predecessors and successors can vary according to the topology. Once a client has received all the weights from the predecessors, it updates its model by averaging the received weights and its own weights. This approach is more effortless to implement compared to Ring-All-Reduce. However, every client requires sufficient bandwidth to exchange its weights with the successors, making this approach a trade-off between RAR and Parameter Server topologies.

This study's Results section (4) demonstrates the effectiveness of a Federated Learning environment with both the RAR paradigm and the Consensus-Based paradigm with 4, 8, and 12 clients interconnected within the same network.

2.2 Model Compression Techniques

This section investigates the possible advantages and drawbacks related to Model Compression Techniques. This set of techniques is fundamental when dealing with

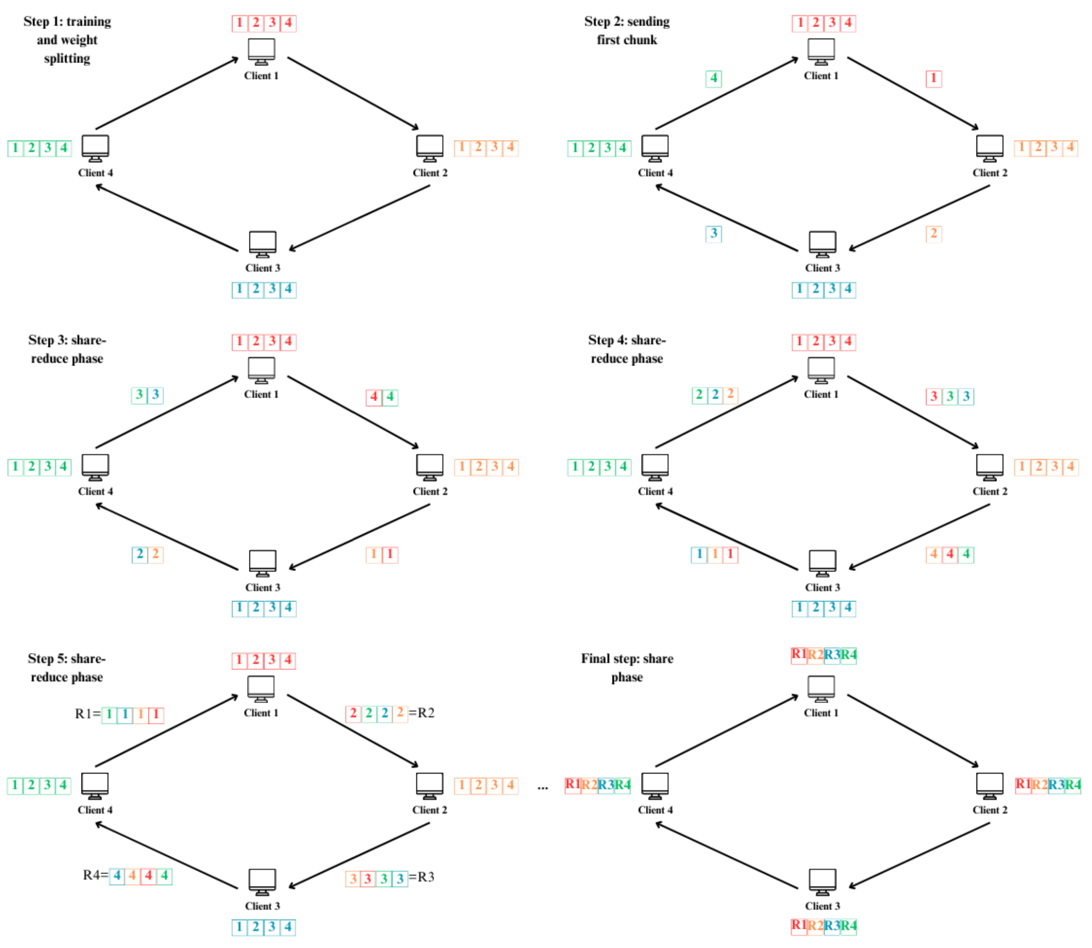

Step 1: training and weight splitting
Step 2: sending first chunk
Step 3: share-reduce phase
Step 4: share-reduce phase
Step 5: share-reduce phase
Final step: share phase
Client 1
Client 2
Client 3
Client 4
R1=
=R2
=R3
R4=
R1 R2 R3 R4

Figure 2.2: Federated Learning process with Ring-All-Reduce strategy and four clients

resource-constrained devices, as it can be exploited to optimize memory usage. In most scenarios, regardless of the Model Compression Technique, optimizing memory usage is linked to loss of accuracy due to model simplification.

Quantization is a powerful Model Compression Technique aimed at weight-precision reduction: the original 32-bit float values are encoded in 8-bit values. This approach decreases memory consumption but is computationally expensive and strongly dependent on hardware.

Low-rank factorization is another Model Compression Technique that exploits a mathematical approach for approximating a dense weight matrix with the product of two lower-dimensional matrices with lower ranks.

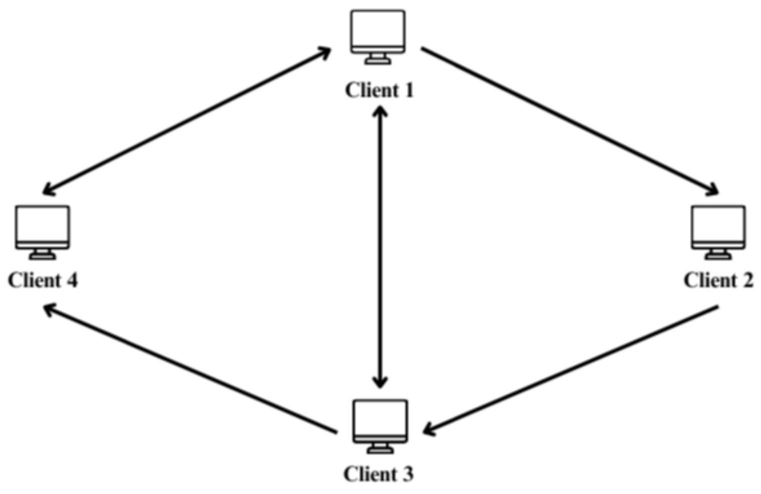


Figure 2.3: Consensus-Based topology with four clients

Knowledge distillation is a complex and effective Model Compression Technique that takes advantage of two different models. A compact model (student) is trained to mimic the outputs of a bigger and more accurate model (teacher); the goal of the student model is to approximate the teacher's model predictions as closely as possible. The teacher model is trained on the original training set, while the student model is trained with the teacher model's predictions. The student model includes the accuracy of its predictions and the similarity of its predictions to those of the teacher model in its loss function.

As stated in [6], there are other Model Compression Techniques, each with different features, pros, and cons. The Model Compression Techniques described in [6] belong to three distinct classes: *Convolution Decomposition*, *Layer Architecture Modification*, and *Weight Compression*.

The class of Convolution Decomposition indicates all techniques aimed at compressing the model's convolutional layers. For example, the Sparse Decomposition of a Convolution Kernel consists of replacing a single computation-intensive convolutional layer with a two-stage decomposition using PCA or multiplication with identity matrices. Direct Sparse Convolution and Separable Depth-wise Convolution are two Model Compression Techniques that also belong to this class.

Regarding the Architecture Modification class, the model can be compressed by replacing the convolutional layers with a fire layer or applying one or more global averaging pooling layers.

The Weight Compression class, instead, includes techniques that remove redundant weights to reduce the size of neural networks. This class's most widely used techniques are *Singular Value Decomposition*, *Sparse Coding*, and *Weight Pruning.* The Singular Value Decomposition (SVD) technique introduces an intermediate layer between two adjacent layers to compress the trained weights and then applies SVD to the weight matrix. The Sparse Coding technique, instead, decomposes the original weight matrix into two matrices using a sparse dictionary learning learned from the original weights.

This work focuses on another Model Compression Technique that belongs to the weight compression class: *Weight Pruning.*

2.2.1 Weight Pruning

Identifying optimal techniques for compressing models by reducing their number of parameters is essential. This reduces memory, battery, and hardware consumption without sacrificing accuracy. In turn, it allows you to deploy lightweight models on resource-constrained devices and guarantee privacy with private on-device computation.

All the pruning techniques studied in this work are implemented within the *PyTorch prune library.* It is important to choose the entity to be pruned to obtain the desired effect; it can be a single layer, a set of layers, or the entire neural network. Supposing to prune a convolutional layer with a pruning percentage equal to 20%, a pruning mask is generated by the selected pruning technique and saved. Finally, pruning is applied prior to each forward step using *PyTorch's forward_pre_hooks.* Specifically, the pruning techniques implemented in *Torch.nn.utils.prune* compute the pruned version of the weight by combining the mask with the original parameters.

The Weight Pruning techniques are classified according to several aspects: random or not, structured or unstructured, local or global. As the name suggests, random pruning selects the weights to be pruned according to the chosen percentage without a specific strategy. This approach may lead to a significant accuracy loss due to the random possibility of pruning relevant weights within the model. The *L1-norm* and the *Ln-norm* are exploited to follow a more precise and controlled approach. These two approaches prune the tensors by removing the specified percentage of weights with the lowest *L1-norm* or *Ln-norm.*

Pruning techniques are also classified between structured and unstructured and differ in how they remove parameters. In structured pruning, parameters are removed in a structured manner, often in groups or patterns. This means the network's neurons, channels, or other structural components are pruned together. Common examples include pruning entire channels in convolutional layers or removing entire neurons in fully connected layers. Structured pruning tends to maintain the network's original shape and connectivity, facilitating hardware implementation and accelerating inference. Unstructured pruning involves removing individual parameters from the neural network without any specific pattern or structure. Each parameter (weight or bias) is independently evaluated, and the least important ones are pruned based on specific criteria, such as magnitude or sensitivity to the output of the model. Unstructured pruning can lead to irregularly shaped networks with sparse connections, which may be less efficient for hardware implementation but can achieve higher compression rates than structured pruning. A comparison between unstructured and structured pruning is illustrated in Figure 2.4.

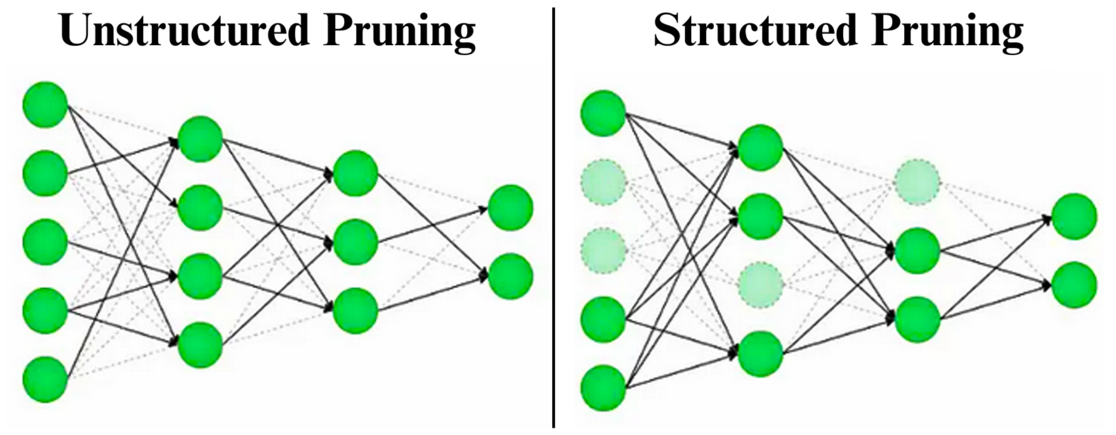


Figure 2.4: Comparison between unstructured pruning and structured pruning

Finally, a pruning technique can be local or global; these two methodologies differ in how weight comparison is made. Local pruning consists of removing a fixed percentage of units/connections from each layer by comparing them within the layer. Global pruning pools all parameters across layers and selects a global fraction to prune.

This study is focused on analyzing three particular Model Compression Techniques that merge the previously described aspects: *Global Unstructured Pruning (GUP)* with L1-norm, *Local Unstructured Pruning (LUP)* with L1-norm, and *Local Structured Pruning (LSP)* with Ln-norm.

Chapter 3

Related work

This chapter introduces several works on Distributed Learning systems and energy consumption analysis.

As a first step, it is fundamental to understand the differences between *Centralized Learning* and *Distributed Learning.* An interesting comparison between the two approaches is illustrated in [7]. The work is focused on performance evaluation within the context of *parallel Stochastic Gradient Descent (SGD).* The results show that decentralized algorithms can achieve comparable or superior results to centralized methods under certain conditions, reduce communication, and increase robustness against failures. These achievements make Decentralized Learning an important alternative in distributed environments.

One of the most challenging aspects of Distributed Learning is represented by handling large datasets. To fully explore this topic, the work [8] investigates dual averaging methods for distributed optimization, focusing on their convergence properties and scalability in network settings. The paper's results indicate that dual averaging can achieve optimal performance while efficiently handling large-scale data.

Among the various approaches of Distributed Learning, *Federated Learning* is the one that is able to enhance privacy during the model weights update. Federated Learning is the core topic of [9]; this work addresses the challenge of traffic prediction in cellular networks using FL, focusing on improving energy efficiency in 5G and 6G networks. The authors followed an energy-aware approach to predict network traffic patterns while reducing overall energy consumption.

A very interesting framework for Federated Learning is described in [10]. It is an open-source Python framework designed to simplify the development and deployment of Decentralized Learning systems. The main advantage of this work is the possibility for researchers and developers to quickly implement and experiment with decentralized Machine Learning algorithms, such as Federated Learning, without requiring in-depth knowledge of the underlying communication protocols and system architecture. Users can seamlessly manage client-server communication, model aggregation, and network topology, making Decentralized Learning more accessible and scalable. This framework inspired us in the implementation of the simulation environment described in 4.1.

The simplest architecture for Distributed Learning is the one with *Parameter Server*. The Parameter Server topology is described in [11]. The proposed Parameter Server framework effectively minimizes communication and, at the same time, ensures rapid convergence of distributed training algorithms. In addition, the framework improves scalability and efficiency in handling large datasets.

This study analyzes several aspects of Federated Learning, including energy consumption. A significant analysis of this topic, particularly the carbon footprint of Distributed and Federated Learning systems, is [12]. This work tackles the following topics: different approaches for model updating to evaluate energy impact, energy consumption for DL and FL scenarios, and the importance of node communication in energy efficiency and carbon footprint. Moreover, it is focused on how network latency and model update frequency affect total energy consumption. The energy efficiency of network architectures and devices participating in Federated Learning can be optimized, for example, by reducing unnecessary communication overhead or compressing updates. Another aspect highlighted by this paper is the importance of managing the carbon footprint in the design of Distributed Machine Learning systems to adopt more sustainable computing practices. The energy cost for learning and communication with Centralized and Federated Learning are illustrated in Figure 3.1.

An interesting strategy to improve energy consumption in Distributed Learning is based on *Model Compression Techniques*. This thesis focuses on Weight Pruning, but several other works have studied this topic. To be more precise, the work [13] develops a model to track the carbon footprint of Federated Learning systems, focusing on the impact of quantization and sparsification techniques on energy consumption. It is demonstrated that these techniques significantly reduce energy consumption and carbon emissions, highlighting their importance for designing sustainable Machine Learning systems, particularly in IoT applications.

An intriguing paper that explores more in detail Model Compression Techniques, communication efficiency, and Federated Learning systems is [14]. It explores the use of quantized neural networks and communication-efficient methods to reduce energy consumption during model training over wireless networks. The usage of quantized neural networks leads to a substantial reduction in energy consumption without significantly impacting the model's performance, which is fundamental for designing sustainable systems.

Among the various Model Compression Techniques, *quantization* is found to be one of the most effective. Model quantization is the main topic of [15]. Quantization techniques based on model equivalence are exploited to reduce the computational

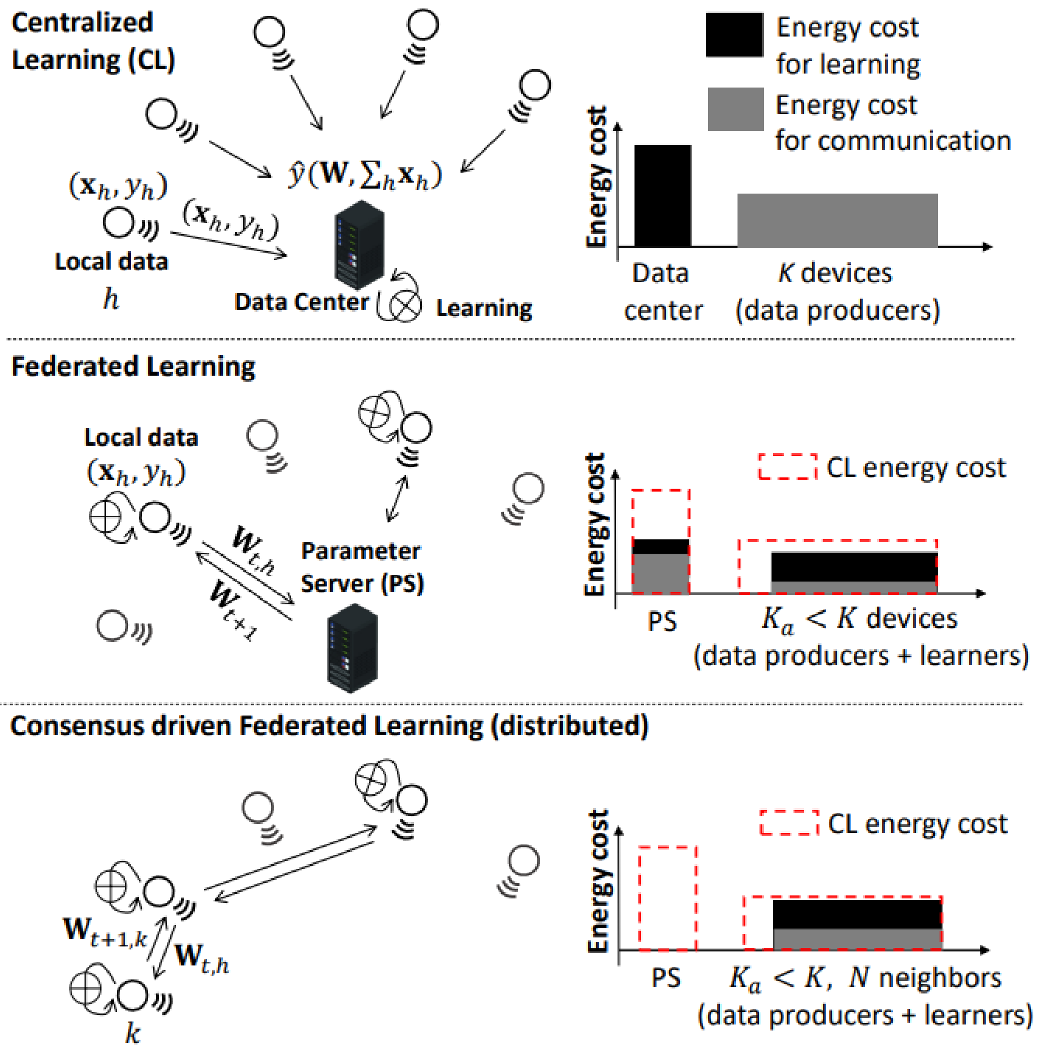


Figure 3.1: Energy cost with different configurations. Image taken from [12]

and memory demands of neural networks. The quantization approach shows that the models can be significantly compressed while maintaining equivalent performance, enabling more efficient use of neural networks in resource-constrained environments.

Similarly, by implementing quantized neural networks, [16] shows that a significant reduction in both energy consumption and communication costs can be achieved within Federated Learning frameworks over cloud-RAN systems.

Model Compression Techniques are particularly effective for developing Machine

Learning strategies on IoT devices. In [17], a compressed *Bayesian Federated Learning* framework has been created for passive radio sensing in industrial IoT contexts. The study shows that communication overhead between devices and the central server in the Federated Learning process can be minimized by compressing updates. This enables reliable sensing in environments with limited bandwidth, while the Bayesian approach improves the robustness of the learning process.

Lastly, improving storage optimization is necessary to develop a Federated Learning environment on IoT devices. An interesting paper on this aspect is [18]; the paper introduces a lossless compression technique for neural network models through exponent sharing. The project's final goal is efficient storage and intelligent computation for CNNs on FPGA systems without losing accuracy.

All these works have provided a solid foundation for the development of this thesis, offering valuable insights, methodologies, and perspectives that have significantly contributed to shaping the direction and approach of the work presented here.

Chapter 4

Results

This chapter presents the results obtained from a federated environment deployed using Docker and Chameleon.

4.1 Simulation environment

The hardware utilized for this study is an RTX 6000 GPU available on Chameleon Cloud. Docker is used to simulate the presence of multiple clients within the same physical device and network. Specifically, many Docker containers are generated as the number of desired clients in the network, allowing each container to replicate the behaviour of an individual client.

The simulations analyze several parameters, including the number of clients in the network (4, 8, and 12), the pruning technique applied (Global Unstructured Pruning, Local Unstructured Pruning, and Local Structured Pruning), the percentage of weights to be pruned (20%, 40%, 60%, or 80%) and the neural network utilized by each client (ResNet18, ResNet50, VGG16, MobileNetV2, TinyYoloV2, and LeNet5). The task of these simulations is Image Classification with the CIFAR10 dataset. The local datasets are derived from the complete CIFAR10 data; in particular, every client applies a different image transformation to the entire dataset. In this way, every client has enough input samples to achieve a good accuracy baseline within an acceptable number of epochs, and, at the same time, every client trains the local model on a unique dataset.

The neural networks employed have different dimensionalities. The LeNet5 is the smallest (548.190 non-zero parameters), while the VGG16 is the biggest (33.630.016 non-zero parameters). The ResNet18 and the ResNet50 have 11.176.832 non-zero parameters and 23.501.951 non-zero parameters, respectively. TinyYoloV2 and MobileNetV2 have 6.303.568 and 2.279.368 non-zero parameters.

The metrics evaluated are accuracy and inference time. Since the network comprises various clients with unique data, the metrics computed for each client vary accordingly. Another critical metric to consider is the bandwidth utilization resulting from clients' communications. The topologies employed are Ring-All-Reduced (RAR) and Consensus-Based (CB).

4.2 Evaluation strategy

Every client initially trains its model for a certain number of epochs, and then the clients perform the Ring-All-Reduce algorithm. In the following step, each client retrains the local model obtained after the RAR or CB algorithm for an additional number of epochs before conducting inference. The accuracy and the inference time recorded after this step are used as baselines for further simulations, and the model obtained is used as the starting model for the next steps. At this point, every client applies the same pruning strategy at a designed percentage to the starting model and then performs inference (Figure 4.1). These final steps are repeated for each pruning strategy and percentage.

This approach allows for an analysis of the impact of a certain pruning technique at a specified percentage on the baselines of both accuracy and inference time.

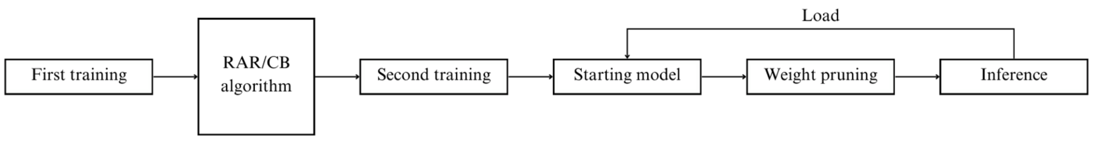


Figure 4.1: The evaluation strategy is employed to assess the impact of various pruning techniques on both accuracy and inference time

4.3 Simulation results

This section shows the results obtained based on the neural network employed.

Each graph in this section contains histograms about accuracy and inference time, and each of them shows a confidence interval of 95%, calculated using the following formula:

$$CI = \bar{x} \pm 1.96 \cdot \frac{\sigma}{\sqrt{n}} \tag{4.1}$$

Where:

- $\bar{x}$ is the mean value of accuracy or inference time collected from all clients in the network,
- σ is the standard deviation of the accuracy or inference time across all clients,
- n is the number of clients used to collect data.

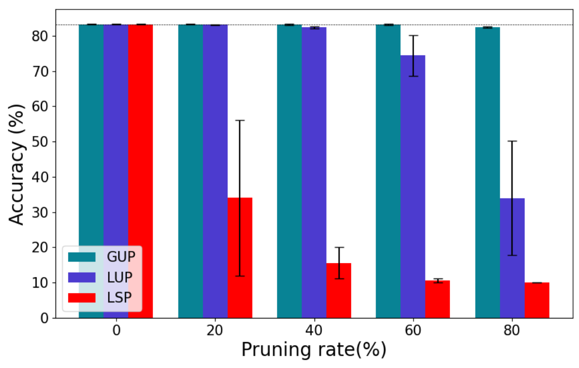
4.3.1 Ring-All-Reduce Topology

ResNet18

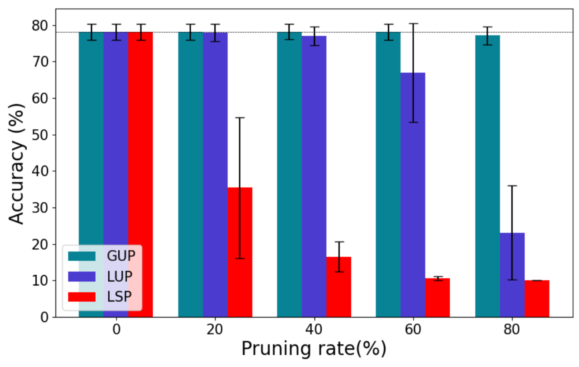
As shown in Figure 4.2, ResNet18 achieves a precision baseline close to 80% regardless of the number of clients within the network. It is evident that the Global Unstructured Pruning technique performs the best compared to the other techniques. The other unstructured technique, LUP, performs similarly to GUP when the pruning percentage is lower than 60%; with a pruning percentage equal to 60% and 80%, LUP's accuracy decreases while the variability increases. The worst pruning technique is Local Structured Pruning. Indeed, the best accuracy value achieved is close to 30%, with a pruning percentage of 20%.

The graphs in Figure 4.3 display the inference times obtained by ResNet18 with 4, 8, and 12 clients. Regardless of the number of clients and the pruning technique, it has been found that the pruning percentage does not particularly influence the inference time's values. In all scenarios, the inference time is slightly lower with respect to the baseline.

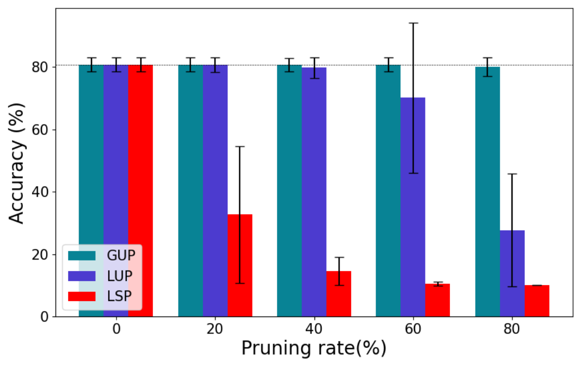
Considering accuracy and inference time trends, Global Unstructured Pruning and Local Unstructured Pruning are the best configurations. In particular, LUP is better than GUP with a percentage lower than 60% since it keeps the accuracy close to the baseline while decreasing the inference time; with a pruning percentage equal to 60% and 80%, GUP has an accuracy closer to the baseline and higher than LUP.



(a) *4 clients*

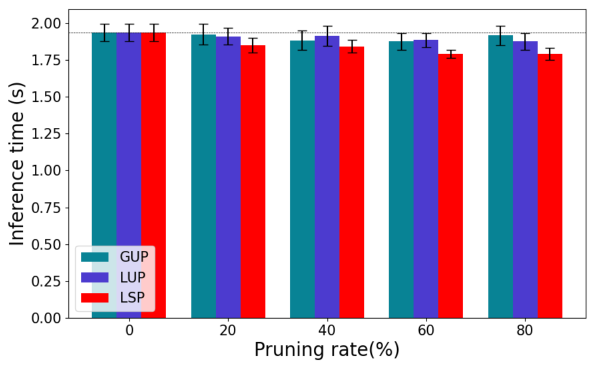


(b) *8 clients*

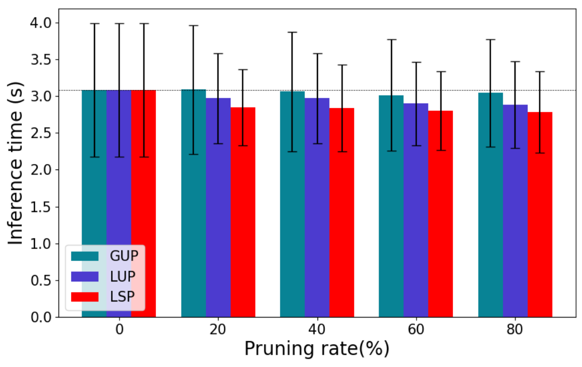


(c) *12 clients*

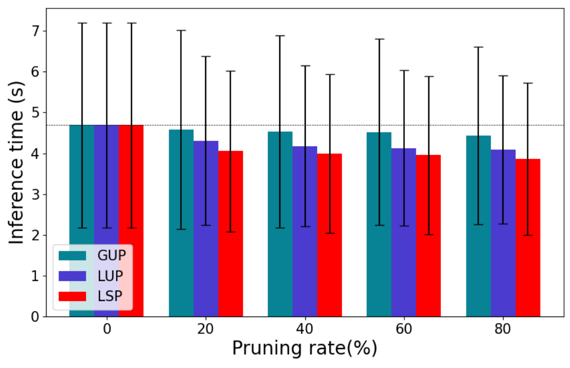
Figure 4.2: Accuracy's behaviour using a ResNet18 with 4, 8, and 12 clients



(a) *4 clients*



(b) *8 clients*



(c) *12 clients*

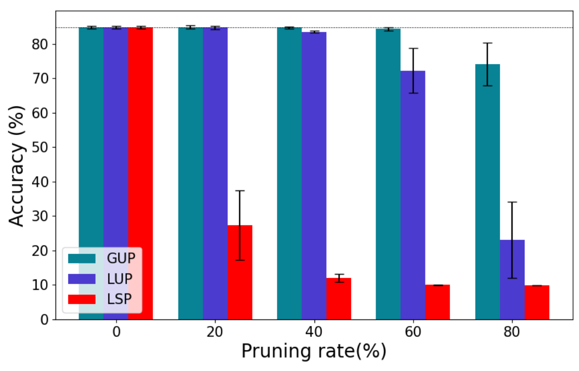
Figure 4.3: Inference time's behaviour using a ResNet18 with 4, 8, and 12 clients

ResNet50

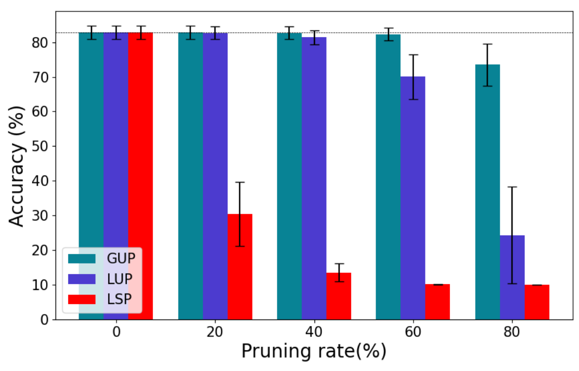
The network exploited in these experiments is the ResNet50, which has twice the number of parameters of ResNet18. This aspect is reflected in the baseline accuracy value higher than in the previous set of experiments, as can be seen in Figure 4.4. The baseline accuracy value is always higher than 80% and decreases when the pruning percentage decreases. The Global Unstructured Pruning and the Local Unstructured Pruning techniques are the closest to the baseline, while the LSP technique is the worst.

The graphs of inference time (Figure 4.5) in this work show that the Global Unstructured Pruning continuously decreases the inference time.

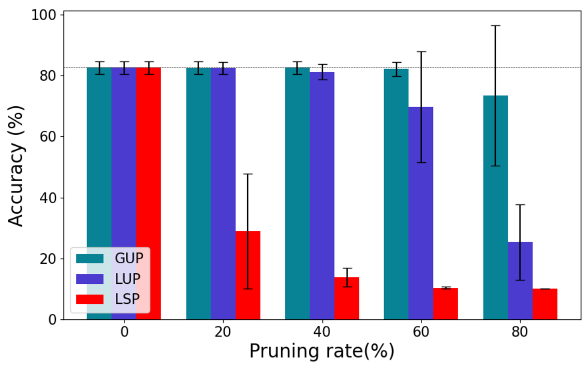
Global Unstructured Pruning is the best technique for this experiment setup because it achieves accuracy close to the baseline while decreasing the inference time. LUP is a good alternative only if the pruning percentage is lower than 60%. LSP decreases inference time but decreases accuracy much more than the other two techniques.



(a) *4 clients*

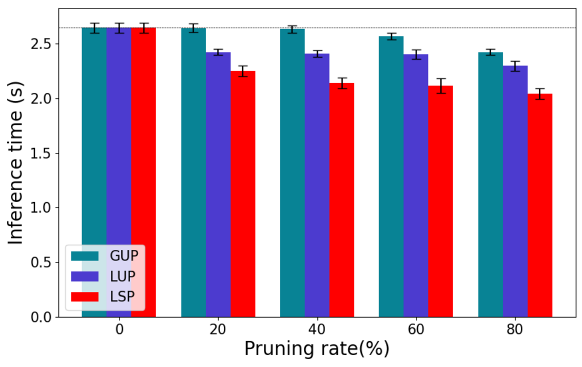


(b) *8 clients*

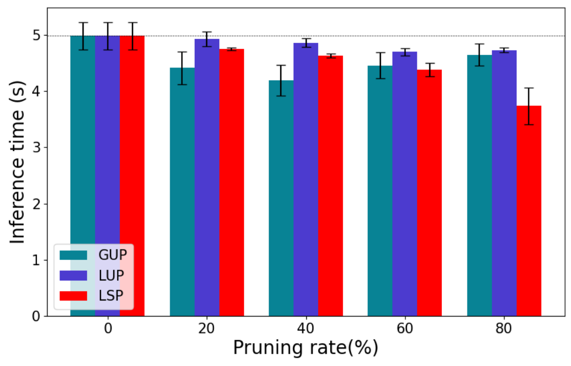


(c) *12 clients*

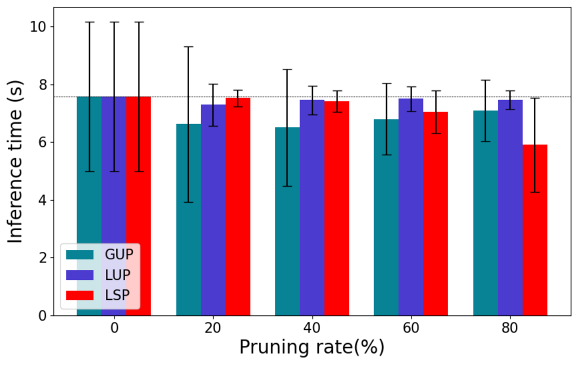
Figure 4.4: Accuracy's behaviour using a ResNet50 with 4, 8, and 12 clients



(a) *4 clients*



(b) *8 clients*



(c) *12 clients*

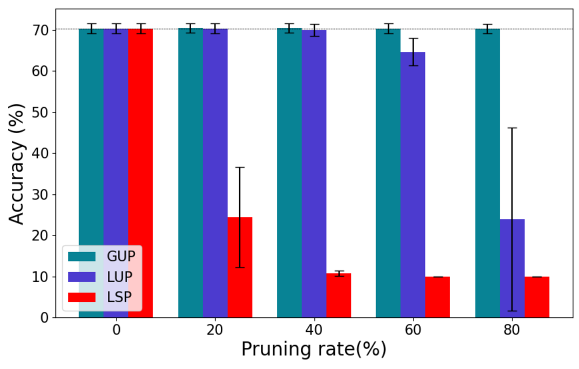
Figure 4.5: Inference time's behaviour using a ResNet50 with 4, 8, and 12 clients

VGG16

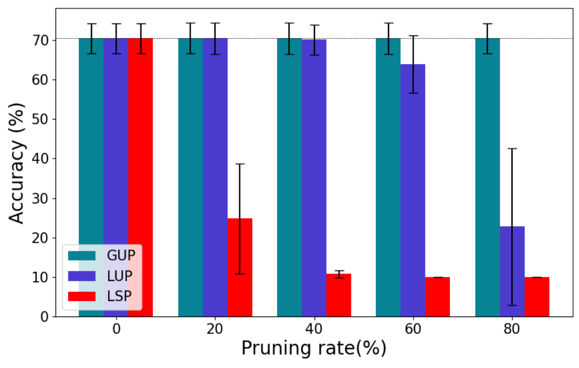
VGG16 is the biggest neural network studied in this work in terms of non-zero parameters. Due to this aspect, the results obtained with the Global Unstrcutrued Pruning technique and shown in Figure 4.6 are close to the baseline regardless of the number of clients. Moreover, there is less variability in accuracy values, meaning that once the model is trained, even an 80% GUP is not able to decrease the model performance. These considerations are not valid for LUP and LSP; Local Unstructured Pruning shows significant variability in accuracy values, particularly with a pruning percentage equal to 80%. LSP, on the other hand, has less variability in results, but the accuracy values are much lower than the baseline.

The inference time's trend shown in Figure 4.7 demonstrates that the Local Structured Pruing technique is the one that leads to the most significant decrease in values. Meanwhile, the GUP strategy generates inference time values comparable to the baseline.

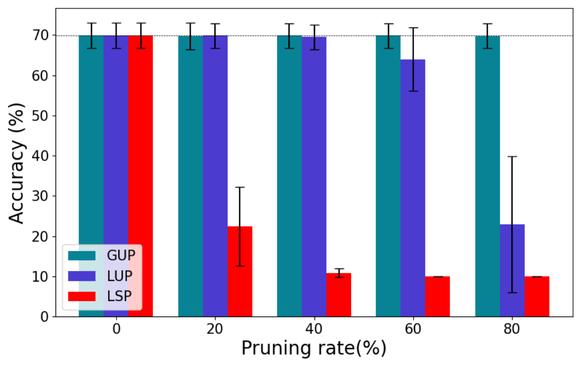
Even if the LSP approach results in lower inference times, it is not the best strategy for this configuration since the accuracy values are too far away from the baseline. LUP is the best strategy to adopt with a pruning percentage lower than 60%, leading to acceptable accuracy values and lower inference times. With higher pruning percentages, Global Unstructured Pruning results in being the best technique in terms of accuracy and inference time.



(a) *4 clients*

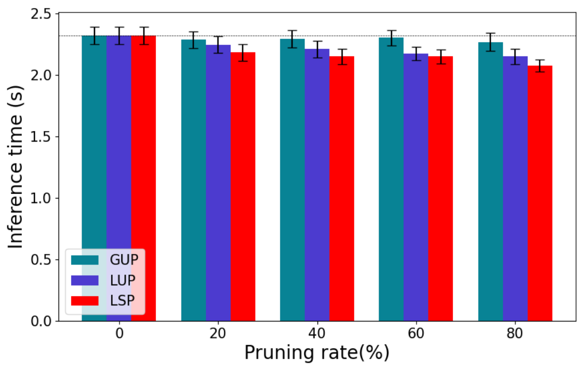


(b) *8 clients*

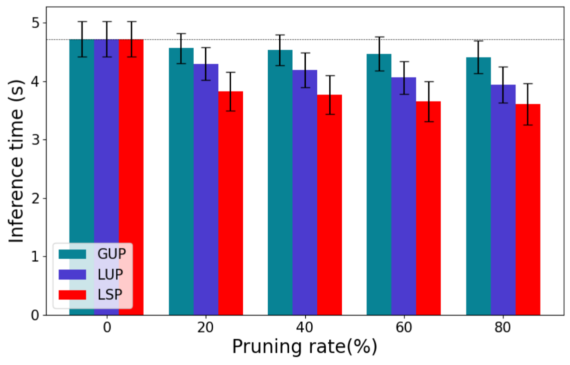


(c) *12 clients*

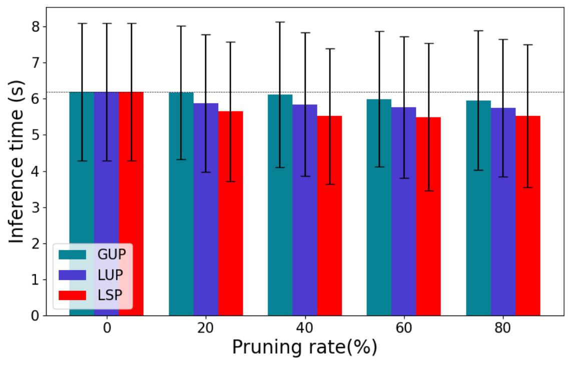
Figure 4.6: Accuracy's behaviour using a VGG16 with 4, 8, and 12 clients



(a) *4 clients*



(b) *8 clients*



(c) *12 clients*

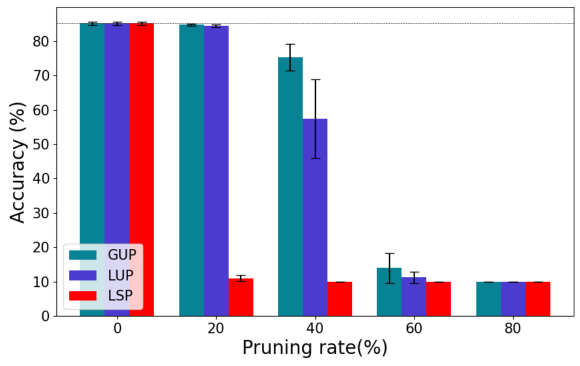
Figure 4.7: Inference time's behaviour using a VGG16 with 4, 8, and 12 clients

MobileNetV2

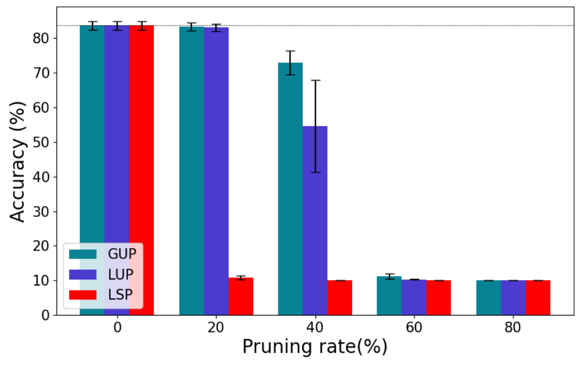
As can be seen in Figure 4.8, all the pruning techniques achieve accuracy values lower than the baseline with a pruning percentage higher than 20%. GUP and LUP perform the same with 20% of pruning percentage, and the accuracy values, in this case, are almost equal to the baseline. Increasing the pruning percentage leads to a very strong decrease in accuracy, which means that the architecture of MobileNetV2 suffers from these kinds of Model Compression Techniques.

This aspect is also highlighted by the graphs in Figure 4.9: the inference time values increase and are higher than the baseline, even if they are supposed to decrease when the pruning percentage increases.

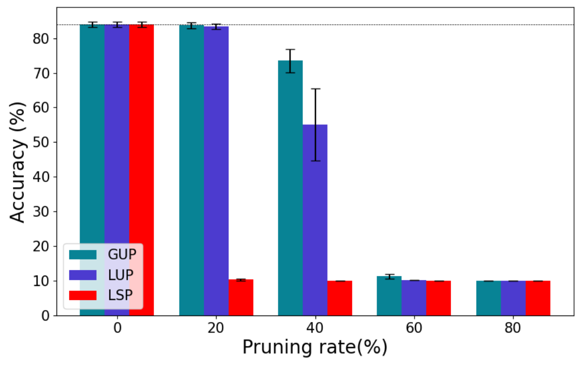
These experiments show that Weight Pruning is not effective with the MobileNetV2 neural network, particularly with high pruning percentages. The performance is worse in terms of both accuracy and inference time compared to the unpruned model.



(a) *4 clients*

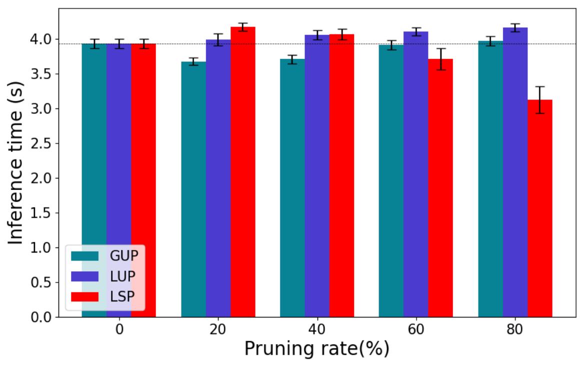


(b) *8 clients*

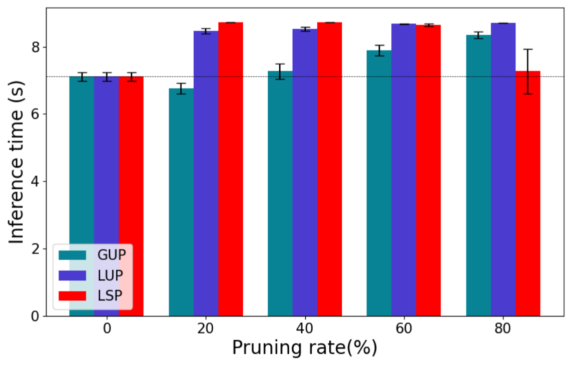


(c) *12 clients*

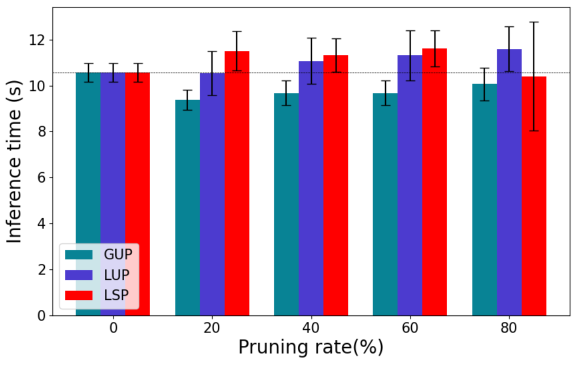
Figure 4.8: Accuracy's behaviour using a MobileNetV2 with 4, 8, and 12 clients



(a) *4 clients*



(b) *8 clients*



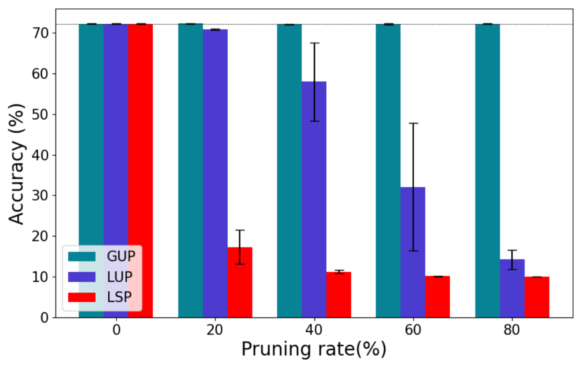
(c) *12 clients*

Figure 4.9: Inference time's behaviour using a MobileNetV2 with 4, 8, and 12 clients

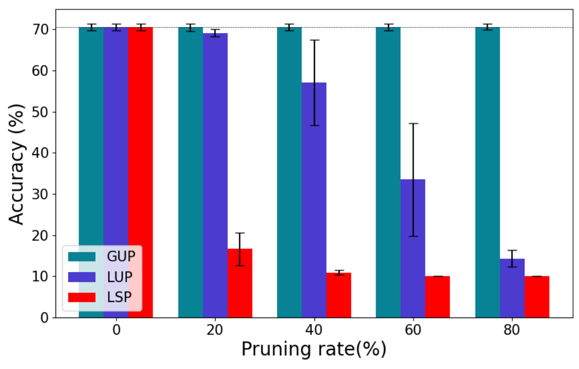
TinyYoloV2

The graphs in Figure 4.10 have an entirely opposite trend with respect to the graphs created with the previous experiment setup. The Global Unstructured Pruning approach keeps the accuracy equal to the baseline regardless of the number of clients or the pruning percentage. Moreover, the values have very low variability. The two local pruning approaches (i.e. LUP and LSP) achieve lower accuracy values; only the LUP approach with 20% of pruning percentage has results comparable with GUP.

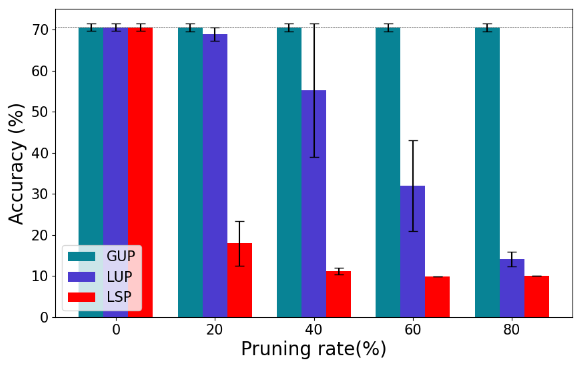
The various pruning methods do not particularly influence the inference time values as seen in Figure 4.11.



(a) *4 clients*

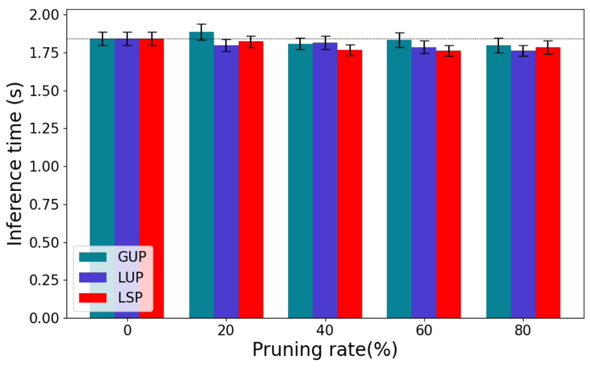


(b) *8 clients*

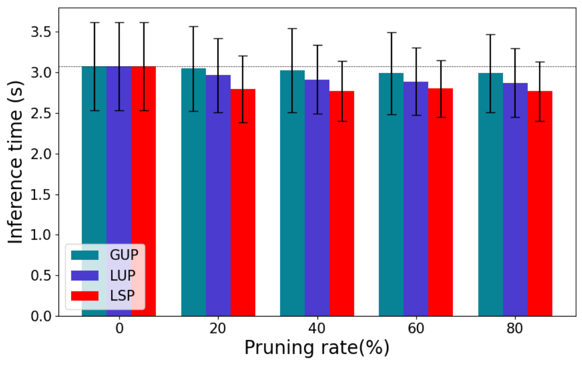


(c) *12 clients*

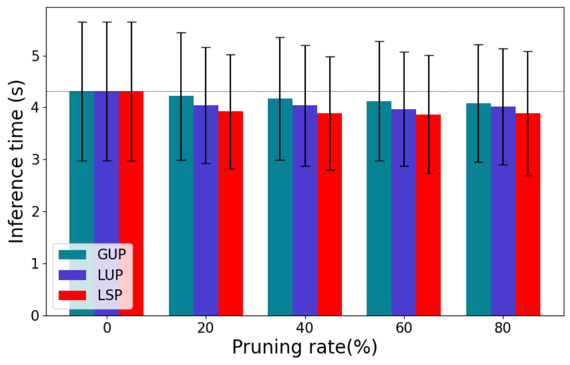
Figure 4.10: Accuracy's behaviour using a TinyYoloV2 with 4, 8, and 12 clients



(a) *4 clients*



(b) *8 clients*



(c) *12 clients*

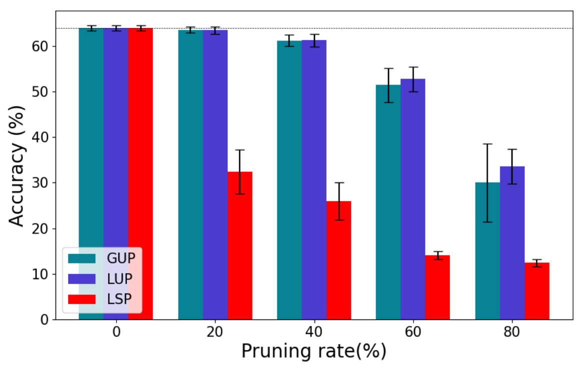
Figure 4.11: Inference time's behaviour using a TinyYoloV2 with 4, 8, and 12 clients

LeNet5

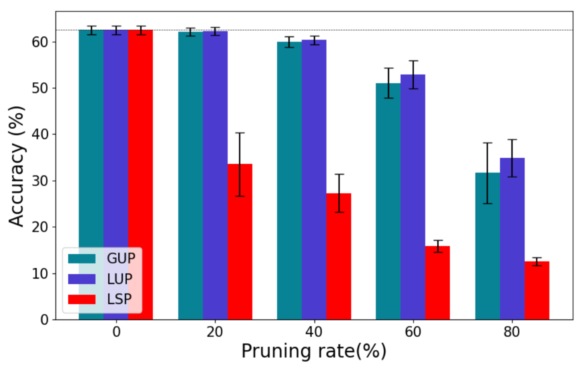
The results obtained with this setup are the ones with the most linear accuracy trend (Figure 4.12). Indeed, the accuracy decreases when the pruning percentage increases, regardless of the number of clients. GUP and LUP are the best approaches since they achieve values closer to the baseline. Local Structured Pruning, in this scenario, too, is the worst approach, but its trend is linear.

Figure 4.13 shows that Weight Pruning does not affect inference time.

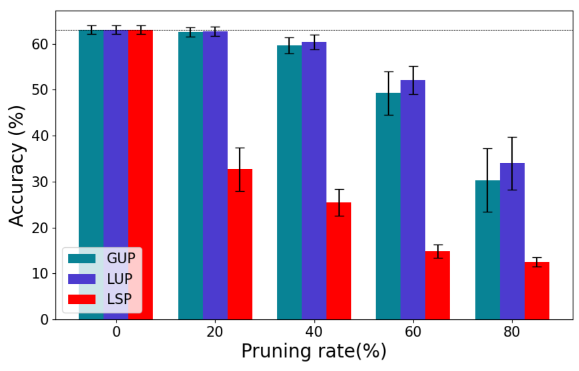
The technique that performs the best with LeNet5 is Local Unstructured Pruning: the accuracy values are better than GUP and LSP, regardless of the pruning percentage and the number of clients.



(a) *4 clients*

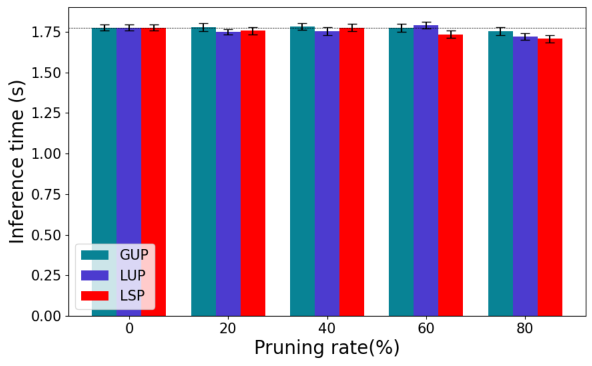


(b) *8 clients*

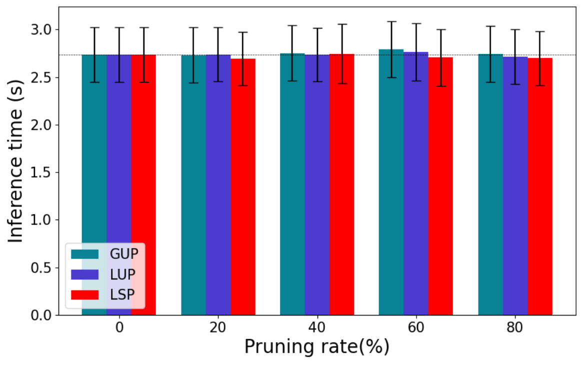


(c) *12 clients*

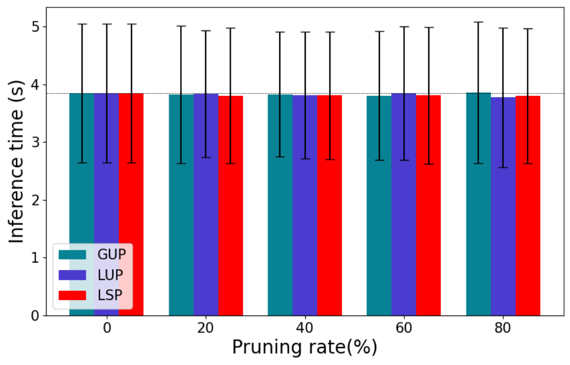
Figure 4.12: Accuracy's behaviour using a LeNet5 with 4, 8, and 12 clients



(a) *4 clients*



(b) *8 clients*



(c) *12 clients*

Figure 4.13: Inference time's behaviour using a LeNet5 with 4, 8, and 12 clients

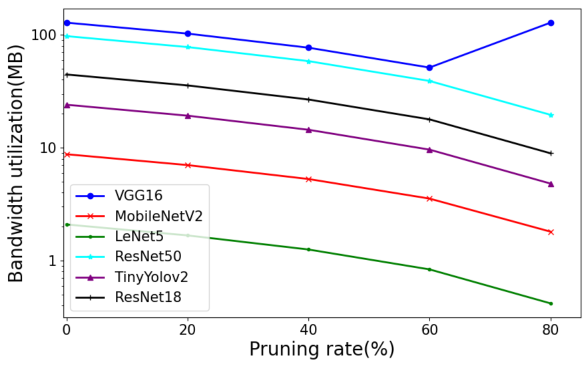
Bandwidth utilization

The aspect related to bandwidth utilization is very important when dealing with specific approaches such as Ring-All-Reduce. Graphs in Figure 4.14 suggest two different approaches to exchange the non-zero parameters of each model across the network.

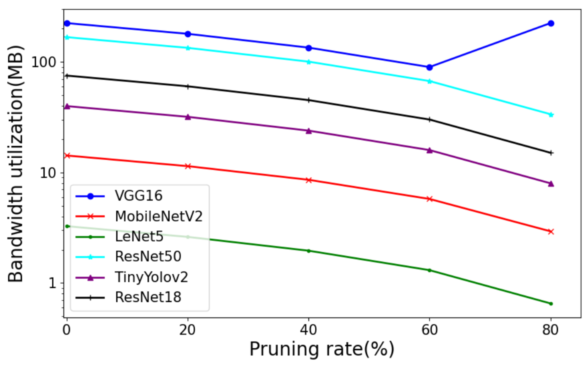
The first strategy exploits the bitmask generated by PyTorch's prune library. After pruning the model, each client sends the non-zero parameters and the bitmask to its successor. In this case, the extra cost due to the transmission of the bitmask is equal to a single bit for each value, but it leads to a higher reconstruction cost. Indeed, when a client receives both the non-zero parameters and the bitmask, it has to iterate over the whole bitmask to understand the correct position of a certain non-zero parameter.

On the other hand, the second approach associates a numerical index to each non-zero parameter. The index indicates the parameter's position inside the model weight matrix, allowing the receiver to immediately understand the correct value's position. The main disadvantage of this approach is that the index needs more than a single bit, particularly if the model is large and has a considerable number of weights, leading to a larger bandwidth consumption.

In other words, the bitmask approach is useful when the main goal is to limit bandwidth utilization, and there are no constraints on each client's reconstruction time. If the goal is to reconstruct the weight matrix received as fast as possible and the communication bandwidth is unlimited, the index approach is preferable.



(a) *Bitmask*



(b) *Index for each model weight*

Figure 4.14: Bandwidth utilization for non-zero parameters exchange with two different approaches

4.3.2 Consensus-Based Topology

The results shown in this section are characterized by a Consensus-Based topology. For this reason, it is essential to know each client how many successors and predecessors they have. A specific setup of clients was chosen to make the results comparable with 4, 8, and 12 clients. Tables 4.1, 4.2, and 4.3 show the number of successors and predecessors of each client for each experiment setup.

Client-id	**#Predecessors**	**#Successors**
Client-1	1	1
Client-2	2	1
Client-3	1	1
Client-4	1	2

Table 4.1: Number of predecessors and successors for each client in a Consensus-Based network with four clients

Client-id	**#Predecessors**	**#Successors**
Client-1	0	4
Client-2	2	2
Client-3	1	2
Client-4	3	3
Client-5	1	2
Client-6	2	1
Client-7	1	1
Client-8	2	2

Table 4.2: Number of predecessors and successors for each client in a Consensus-Based network with eight clients

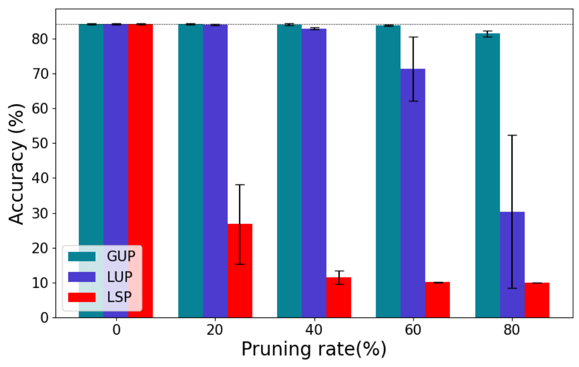
Client-id	**#Predecessors**	**#Successors**
Client-1	0	5
Client-2	1	6
Client-3	1	7
Client-4	1	5
Client-5	2	3
Client-6	3	2
Client-7	2	3
Client-8	3	2
Client-9	1	2
Client-10	4	1
Client-11	4	1
Client-12	6	0

Table 4.3: Number of predecessors and successors for each client in a Consensus-Based network with twelve clients

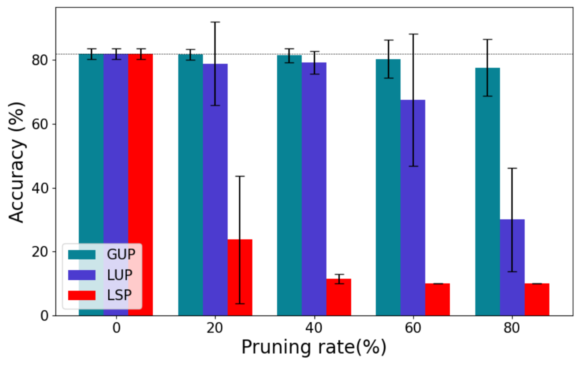
ResNet18

The results displayed in Figure 4.15 show that the Global Unstructured Pruning technique achieves the best results compared to the other pruning techniques. The values obtained with LUP and LSP are characterized by high variability and are always lower than the baseline.

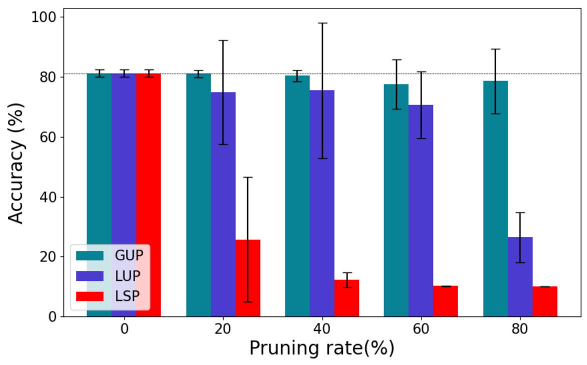
The pruning techniques do not appear to particularly affect the inference time values, but LSP decreases them the most.



(a) *4 clients*

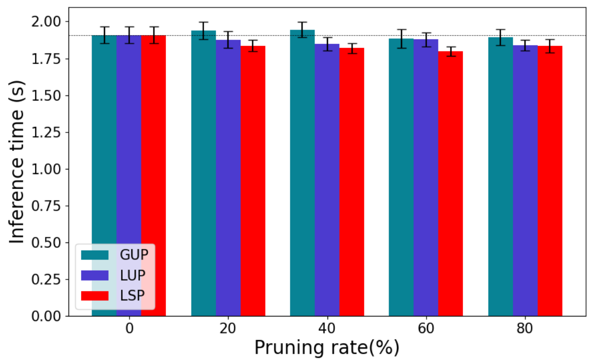


(b) *8 clients*

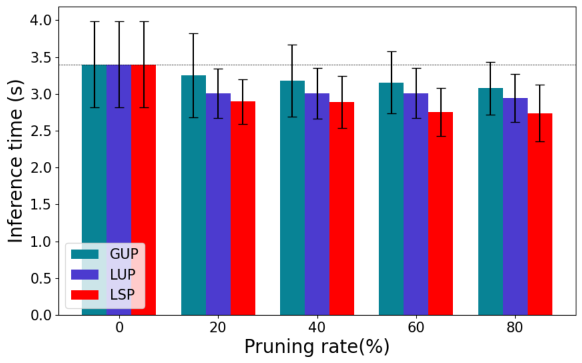


(c) *12 clients*

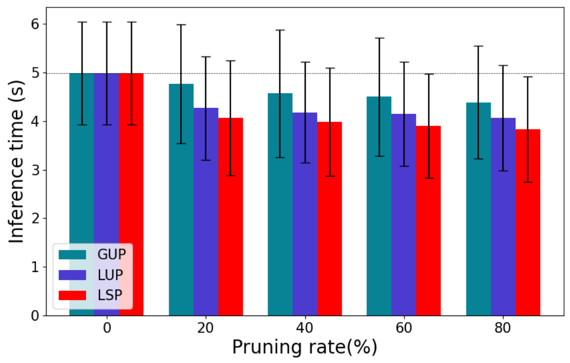
Figure 4.15: Accuracy's behaviour using a ResNet18 with 4, 8, and 12 clients



(a) *4 clients*



(b) *8 clients*



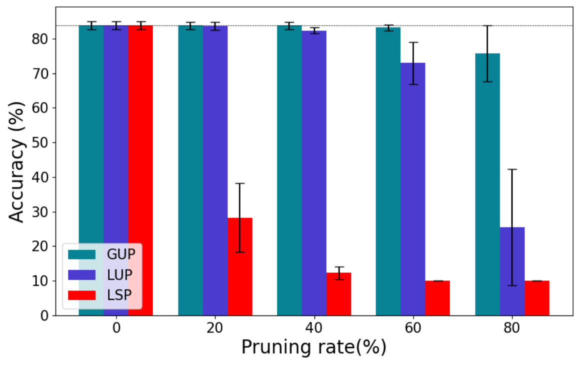
(c) *12 clients*

Figure 4.16: Inference time's behaviour using a ResNet18 with 4, 8, and 12 clients

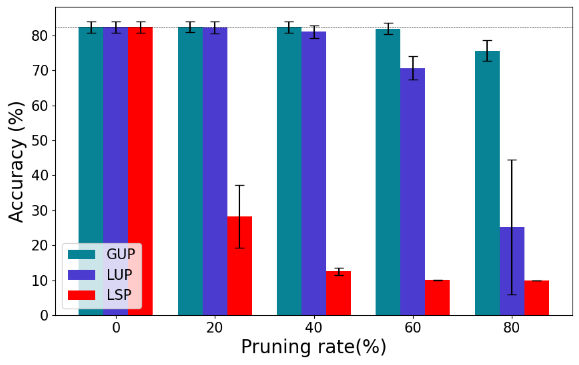
ResNet50

The results in Figure 4.17 confirm the trend highlighted with ResNet18, but the values have less variability due to the larger model size. Global Unstructured Pruning confirms itself as the best Weight Pruning technique, able to achieve values similar to the baseline even when the pruning percentage is higher than 60%.

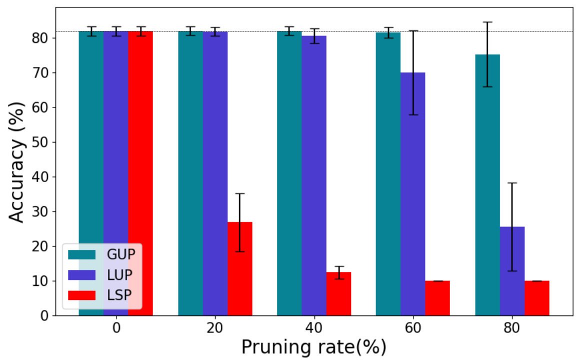
Inference time values (Figure 4.18) are lower than the baseline, in particular with LSP, and their variability is very low.



(a) *4 clients*

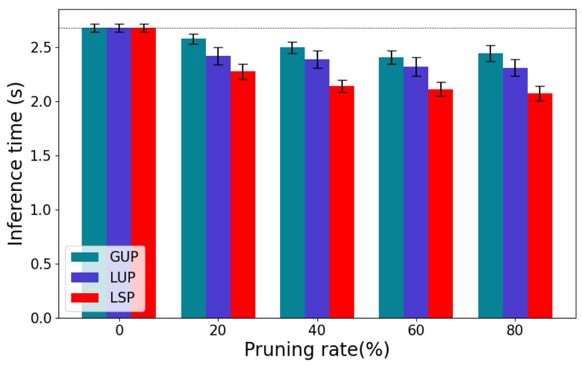


(b) *8 clients*

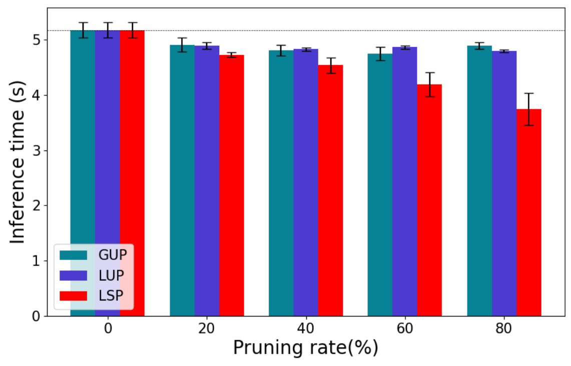


(c) *12 clients*

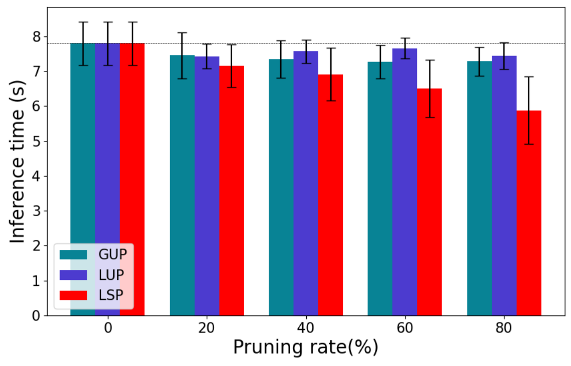
Figure 4.17: Accuracy's behaviour using a ResNet50 with 4, 8, and 12 clients



(a) *4 clients*



(b) *8 clients*



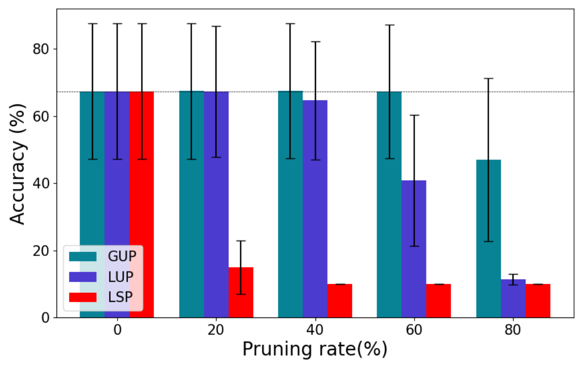
(c) *12 clients*

Figure 4.18: Inference time's behaviour using a ResNet50 with 4, 8, and 12 clients

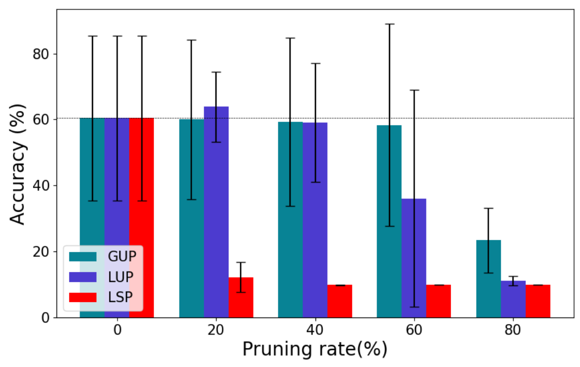
VGG16

Due to the large size of the model, the accuracy values in Figure 4.19 are very unstable and have a large variability. It is hard to indicate which pruning technique performs the best, while it is very easy to understand that LSP is the worst.

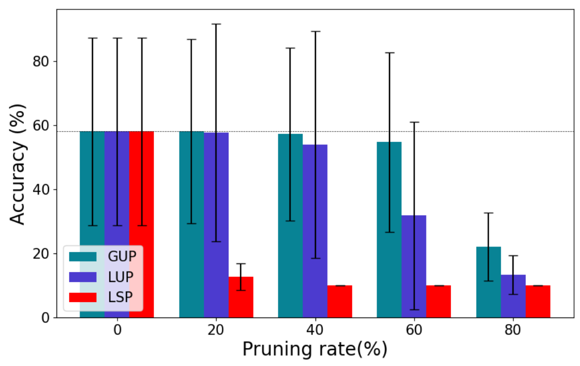
Inference time values have a variability that increases when the number of clients increases, but the results obtained are similar to the baseline in all scenarios. The inference values behaviour can be seen in Figure 4.20.



(a) *4 clients*

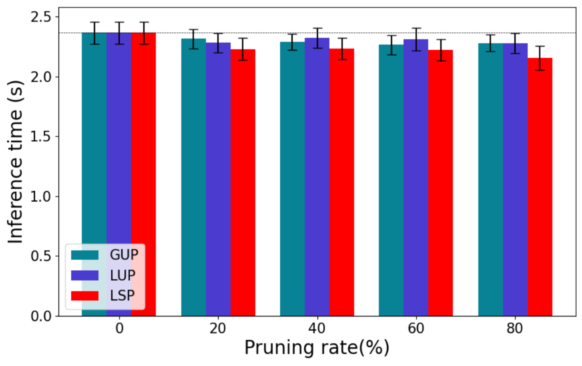


(b) *8 clients*

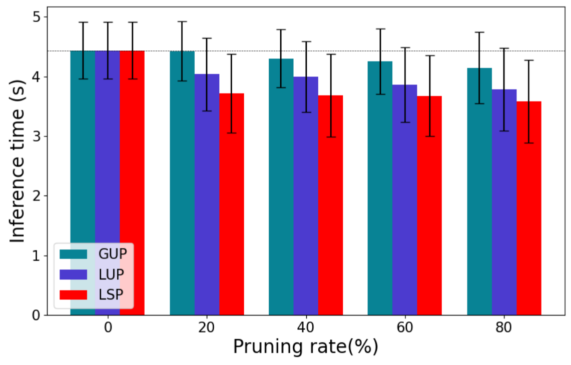


(c) *12 clients*

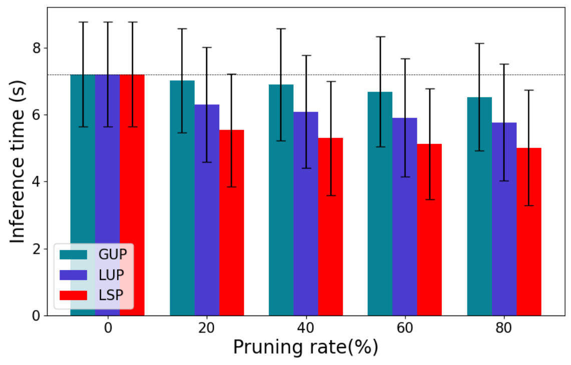
Figure 4.19: Accuracy's behaviour using a VGG16 with 4, 8, and 12 clients



(a) *4 clients*



(b) *8 clients*



(c) *12 clients*

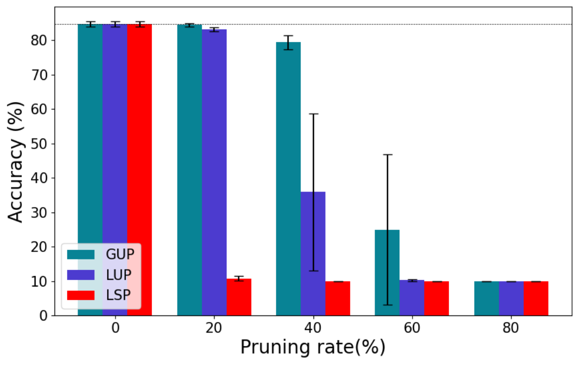
Figure 4.20: Inference time's behaviour using a VGG16 with 4, 8, and 12 clients

MobileNetV2

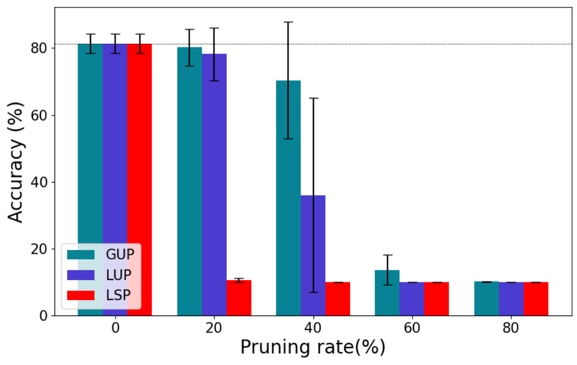
The graphs in Figure 4.19 show that the accuracy values are affected by a significant decrease when the pruning technique is equal to 60% or 80%. Global Unstructured Pruning is the only technique that can maintain a performance similar to the baseline, but only with 20% Weight Pruning.

Furthermore, GUP is the only technique able to decrease the inference time regardless of the number of clients, as shown in Figure 4.22.

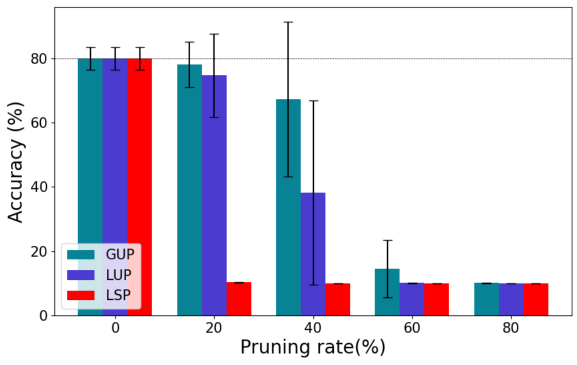
With this experiment setup, Global Unstructured Pruning is clearly the best approach to keep an acceptable accuracy value while decreasing inference time and model size.



(a) *4 clients*

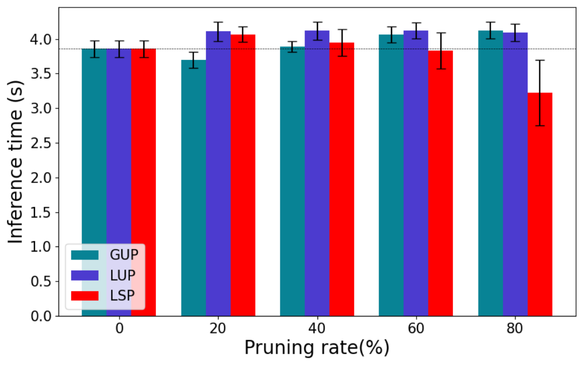


(b) *8 clients*

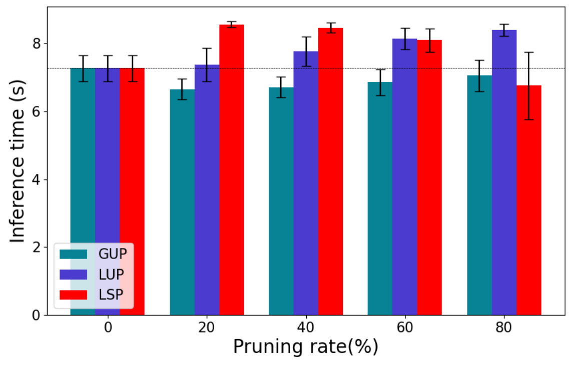


(c) *12 clients*

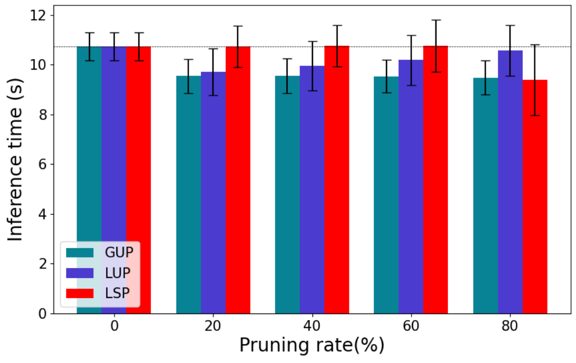
Figure 4.21: Accuracy's behaviour using a MobileNetV2 with 4, 8, and 12 clients



(a) *4 clients*



(b) *8 clients*



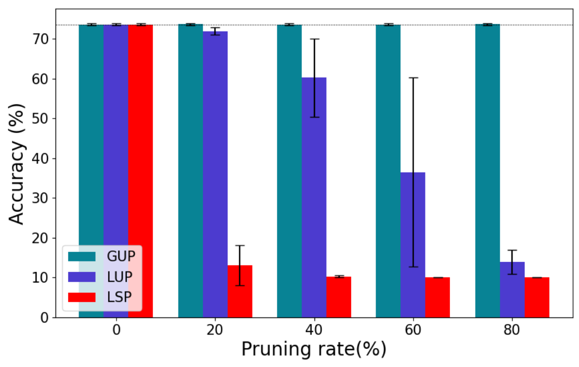
(c) *12 clients*

Figure 4.22: Inference time's behaviour using a MobileNetV2 with 4, 8, and 12 clients

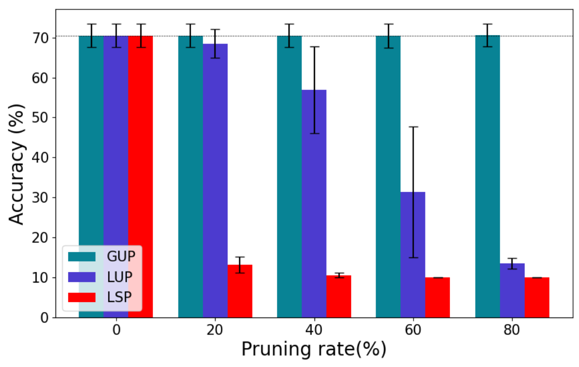
TinyYoloV2

With this neural network, Global Unstructured Pruning achieves the same accuracy as the baseline even with 80% of pruning, while LUP and LSP perform worse (Figure 4.23).

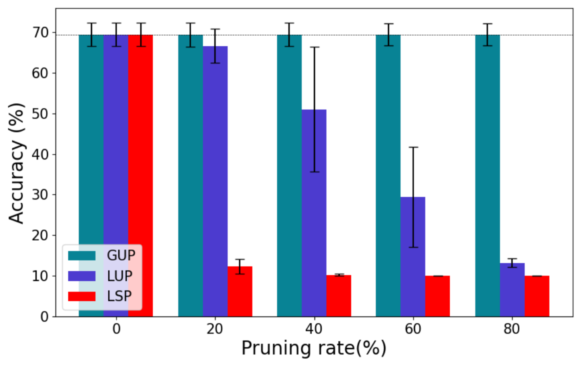
GUP keeps the inference time similar to or lower than the baseline, but LUP and LSP are the techniques that reduce it the most.



(a) *4 clients*

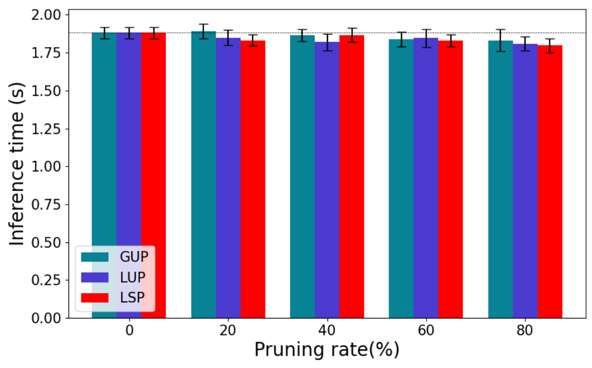


(b) *8 clients*

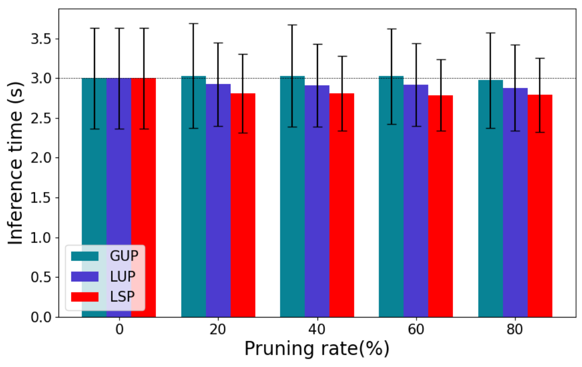


(c) *12 clients*

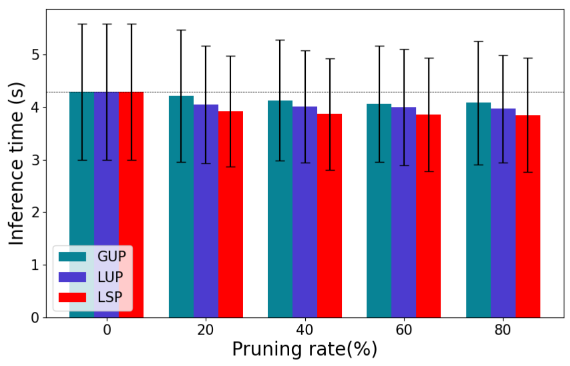
Figure 4.23: Accuracy's behaviour using a TinyYoloV2 with 4, 8, and 12 clients



(a) *4 clients*



(b) *8 clients*



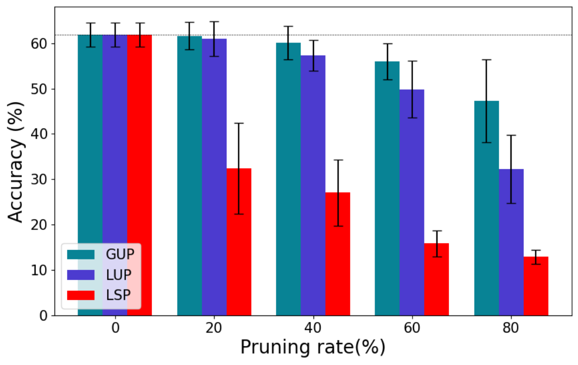
(c) *12 clients*

Figure 4.24: Inference time's behaviour using a TinyYoloV2 with 4, 8, and 12 clients

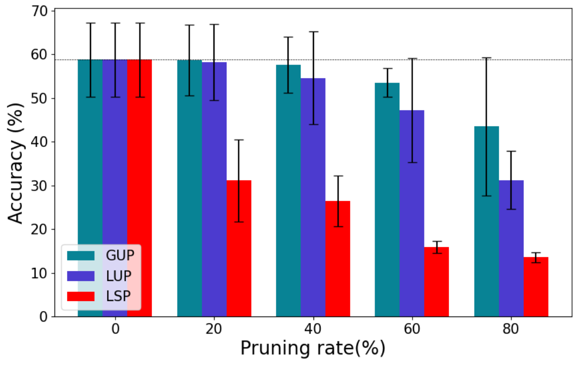
LeNet5

Figure 4.25 shows a linear trend with low variability, and this is due to the small size of the model used. The baseline accuracy is not extremely high, but GUP is the only technique to mimic its behaviour when the pruning percentage is 20%.

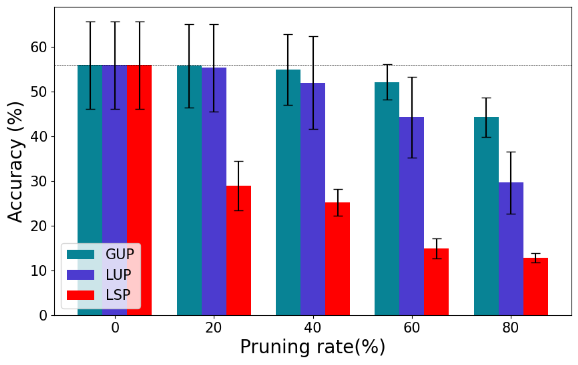
The inference time values obtained (shown in Figure 4.26) always behave the same regardless of the number of clients, the percentage of pruning or the pruning technique.



(a) *4 clients*

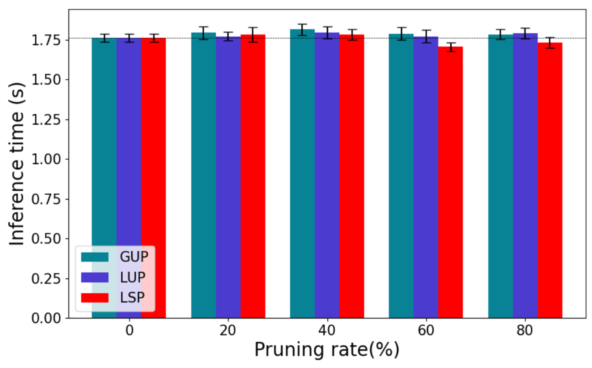


(b) *8 clients*

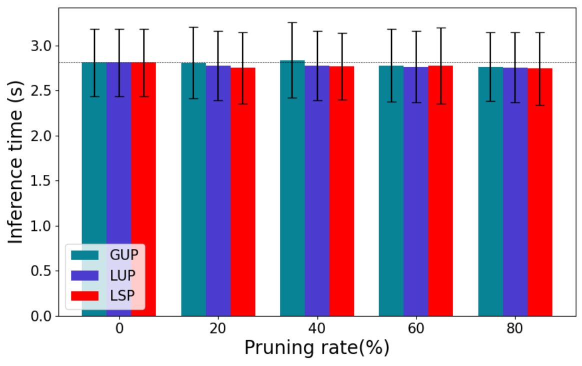


(c) *12 clients*

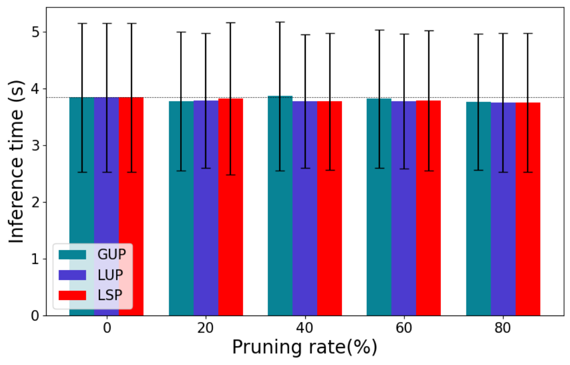
Figure 4.25: Accuracy's behaviour using a LeNet5 with 4, 8, and 12 clients



(a) *4 clients*



(b) *8 clients*



(c) *12 clients*

Figure 4.26: Inference time's behaviour using a LeNet5 with 4, 8, and 12 clients

Memory footprint

Table 4.4 shows the size of the files where different models are saved. The PyTorch format (i.e., `.pt`) has been analyzed, in particular, the models have been stored by exploiting sparse tensors. The pruning approach leads to weight matrices with several zero values, and for this reason, to highlight the memory impact of pruning, it is fundamental to use sparse tensors. As seen in Table 4.4, pruning can reduce the memory footprint, so it is possible to store pruned models using smaller size files while performing as shown in 4.

Model name	**Unpruned**	**20%**	**40%**	**60%**	**80%**
ResNet18	393.45 MB	313.60 MB	233.79 MB	157.48 MB	78.80 MB
ResNet50	845.73 MB	672.61 MB	499.80 MB	339.01 MB	170.03 MB
VGG16	866.22 MB	693.41 MB	531.78 MB	346.73 MB	173.55 MB
MobileNetV2	78.46 MB	62.88 MB	47.30 MB	31.10 MB	16.04 MB
TinyYoloV2	216.31 MB	173.07 MB	129.83 MB	86.60 MB	43.36 MB
LeNet5	10.82 MB	8.68 MB	6.50 MB	4.34 MB	2.27 MB

Table 4.4: Comparison of the file sizes of `.pt` model weight files across different neural network architectures

Chapter 5

Conclusion

This final section presents the main results achieved in this work and analyzes future improvements.

This work aims to understand the effect of several Model Compression Techniques in a Federated environment with different topologies and numbers of clients. More in detail, Global Unstructured Pruning, Local Unstructured Pruning, and Local Structured Pruning have been analyzed with Ring-All-Reduce and Consensus-Based topologies with 4, 8, and 12 clients. The metrics evaluated are accuracy, inference time, and bandwidth utilization.

This study's final goal is to find the best resource-constrained device approach. For this reason, it is fundamental to discover a solution that optimizes storage consumption and bandwidth utilization without decreasing precision.

The Results section (4) highlights several interesting behaviours depending on the pruning percentage and technique.

In most scenarios, Global Unstructured Pruning represents the best approach in terms of accuracy, mainly when the pruning percentage is higher than 40%. On the other hand, Local Unstructured Pruning achieves better performance than GUP when the pruning percentage is equal to 20% or 40%. In this case, the pruned model achieves higher accuracy values while decreasing the inference time. A common aspect that characterizes all the experiments is the poor performance obtained by Local Structured Pruning. This technique achieves shallow accuracy values and never improves the inference time.

It is interesting to study the trend of MobileNetV2 accuracy: a pruning percentage equal to 20% is nearly imperceptible in terms of values, but with higher percentages, precision drops with each pruning technique. TinyYoloV2, instead, has the opposite behaviour: even with 80% of Global Unstructured Pruning, the accuracy value does not change from the baseline.

Regarding the topology to exploit, the Ring-All-Reduce and the Consensus-Based topology achieve the same accuracy results. The RAR topology enhances privacy and improves the device's local memory footprint, and for this reason, it is preferable to the Consensus-Based topology.

The Ring-All-Reduce approach presents another essential metric to be evaluated: bandwidth utilization. The experiments demonstrate the possibility of reducing the amount of data shared across the network by sending only weights different from zero.

Another fundamental aspect is related to the memory footprint. The experimental results show that every pruning technique decreases the file size in which the model is saved according to the exploited pruning percentage. By saving only the weights that are not pruned, it is possible to decrease the .pt file size, which describes the model and permits the finding of a tradeoff between the memory consumption and the accuracy values: the strategy can be implemented on resource-constrained devices by pruning the model, although, in most of the scenarios, the accuracy decreases.

In conclusion, Global and Local Unstructured Pruning techniques perform the best. Their behaviours differ depending on the pruning percentage and the neural network exploited. Pruning is also particularly useful for decreasing the model's memory footprint, allowing the development of Image Classification tasks on memory-constrained devices. The adopted topology does not have a particularly influential effect on the accuracy or inference time values, but the RAR topology can improve privacy and memory impact.

List of Tables

List of Figures

Bibliography

[1] Transforma Insights. «Current IoT Forecast Highlights». In: *Transforma Insights* 31.07 (2023) (cit. on p. 1).

[2] Joost Verbraeken, Matthijs Wolting, Jonathan Katzy, Jeroen Kloppenburg, Tim Verbelen, and Jan S. Rellermeyer. «A Survey on Distributed Machine Learning». In: *ACM Comput. Surv.* 53.2 (Mar. 2020). ISSN: 0360-0300. DOI: 10.1145/3377454. URL: https://doi.org/10.1145/3377454 (cit. on p. 4).

[3] Zongshun Zhang, Andrea Pinto, Valeria Turina, Flavio Esposito, and Ibrahim Matta. «Privacy and Efficiency of Communications in Federated Split Learning». In: *IEEE Transactions on Big Data* 9.5 (2023), pp. 1380–1391. DOI: 10.1109/TBDATA.2023.3280405 (cit. on p. 4).

[4] Jie Wen, Zhixia Zhang, Yang Lan, Zhihua Cui, Jianghui Cai, and Wensheng Zhang. «A survey on federated learning: challenges and applications». In: *International Journal of Machine Learning and Cybernetics* 14.2 (2023), pp. 513–535 (cit. on p. 5).

[5] Giovanni Neglia, Chuan Xu, Don Towsley, and Gianmarco Calbi. «Decentralized gradient methods: does topology matter?» In: *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*. Ed. by Silvia Chiappa and Roberto Calandra. Vol. 108. Proceedings of Machine Learning Research. PMLR, 26–28 Aug 2020, pp. 2348–2358. URL: https://proceedings.mlr.press/v108/neglia20a.html (cit. on p. 5).

[6] Kaiming Nan, Sicong Liu, Junzhao Du, and Hui Liu. «Deep model compression for mobile platforms: A survey». In: *Tsinghua Science and Technology* 24.6 (2019), pp. 677–693. DOI: 10.26599/TST.2018.9010103 (cit. on p. 9).

[7] Xiangru Lian, Ce Zhang, Huan Zhang, Cho-Jui Hsieh, Wei Zhang, and Ji Liu. «Can decentralized algorithms outperform centralized algorithms? a case study for decentralized parallel stochastic gradient descent». In: *Advances in neural information processing systems* 30 (2017) (cit. on p. 12).

[8] J. C. Duchi, A. Agarwal, and M. J. Wainwright. «Dual Averaging for Distributed Optimization: Convergence Analysis and Network Scaling». In: *IEEE Transactions on Automatic Control* 57.3 (Mar. 2012), pp. 592–606. ISSN: 1558-2523. DOI: `10.1109/tac.2011.2161027`. URL: `http://dx.doi.org/10.1109/TAC.2011.2161027` (cit. on p. 12).

[9] Vasileios Perifanis, Nikolaos Pavlidis, Selim F Yilmaz, Francesc Wilhelmi, Elia Guerra, Marco Miozzo, Pavlos S Efraimidis, Paolo Dini, and Remous-Aris Koutsiamanis. «Towards energy-aware federated traffic prediction for cellular networks». In: *2023 Eighth International Conference on Fog and Mobile Edge Computing (FMEC)*. IEEE. 2023, pp. 93–100 (cit. on p. 12).

[10] Akash Dhasade, Anne-Marie Kermarrec, Rafael Pires, Rishi Sharma, and Milos Vujasinovic. «Decentralized learning made easy with DecentralizePy». In: *Proceedings of the 3rd Workshop on Machine Learning and Systems*. 2023, pp. 34–41 (cit. on p. 12).

[11] Mu Li, David G Andersen, Alexander J Smola, and Kai Yu. «Communication efficient distributed machine learning with the parameter server». In: *Advances in Neural Information Processing Systems* 27 (2014) (cit. on p. 13).

[12] Stefano Savazzi, Vittorio Rampa, Sanaz Kianoush, and Mehdi Bennis. «An energy and carbon footprint analysis of distributed and federated learning». In: *IEEE Transactions on Green Communications and Networking* 7.1 (2022), pp. 248–264 (cit. on pp. 13, 14).

[13] Luca Barbieri, Stefano Savazzi, Sanaz Kianoush, Monica Nicoli, and Luigi Serio. «A carbon tracking model for federated learning: Impact of quantization and sparsification». In: *2023 IEEE 28th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)*. IEEE. 2023, pp. 213–218 (cit. on p. 13).

[14] Minsu Kim, Walid Saad, Mohammad Mozaffari, and Merouane Debbah. «Green, quantized federated learning over wireless networks: An energy-efficient design». In: *IEEE transactions on wireless communications* (2023) (cit. on p. 13).

[15] Huiqu Yang, Jian Xu, Guowei Yang, Ming Zhang, and Hong Qin. «Neural Network Quantization Based on Model Equivalence». In: *2022 International Conference on High Performance Big Data and Intelligent Systems (HDIS)*. IEEE. 2022, pp. 8–12 (cit. on p. 13).

[16] Jiali Wang, Yijie Mao, Ting Wang, and Yuanming Shi. «Green federated learning over cloud-ran with limited fronthual capacity and quantized neural networks». In: *IEEE Transactions on Wireless Communications* (2023) (cit. on p. 14).

[17] Luca Barbieri, Stefano Savazzi, and Monica Nicoli. «Compressed Bayesian Federated Learning for Reliable Passive Radio Sensing in Industrial IoT». In: *arXiv preprint arXiv:2405.05855* (2024) (cit. on p. 15).

[18] Prachi Kashikar, Olivier Sentieys, and Sharad Sinha. «Lossless Neural Network Model Compression Through Exponent Sharing». In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* (2023) (cit. on p. 15).