Federated Learning for edge computing: Real-Time Object Detection

Ardit Memia

Supervisor: Addi Ait-Mlouk
Examiner: Göran Falkman

Master Degree Project (15 ECTS) in Informatics with a specialization in Data Science
Privacy, Information and Cyber Security
Spring term 2023
ABSTRACT

In domains where data is sensitive or private, there is a great value in methods that can learn in a distributed manner without the data ever leaving the local devices. Federated Learning (FL) has recently emerged as a promising solution to collaborative machine learning challenges while maintaining data privacy. With FL, multiple entities, whether cross-device or cross-silo, can jointly train models without compromising the locality or privacy of their data. Instead of moving data to a central storage system or cloud for model training, code is moved to the data owners’ local sites, and incremental local updates are combined into a global model. In this way FL enhances data privacy and reduces the probability of eavesdropping to a certain extent.

In this thesis we have utilized the means of Federated Learning into a Real-Time Object Detection (RTOB) model in order to investigate its performance and privacy awareness towards a traditional centralized ML environment. Several object detection models have been built using YOLO framework and training with a custom dataset for indoor object detection. Local tests have been performed and the most optimal model has been chosen by evaluating training and testing metrics and afterwards using NVIDIA Jetson Nano external device to train the model and integrate into a Federated Learning environment using an open-source FL framework. Experiments has been conducted through the path in order to choose the optimal YOLO model (YOLOv8) and the best fitted FL framework to our study (FEDn).

We observed a gradual enhancement in balancing the APC factors (Accuracy-Privacy-Communication) as we transitioned from basic local models to the YOLOv8 implementation within the FEDn system, both locally and on the SSC Cloud production environment. Although we encountered technical challenges deploying the YOLOv8-FEDn system on the SSC Cloud, preventing it from reaching a finalized state, our preliminary findings indicate its potential as a robust foundation for FL applications in RTOB models at the edge.

Keywords: Federated Learning, Artificial Intelligence, Machine Learning, Edge computing, Object Detection, Decentralized AI.
Table of Contents

1. Introduction .................................................................................................................. 5
   1.1. Problem Definition ................................................................................................. 6
   1.2. Content outline ....................................................................................................... 7

2. Background .................................................................................................................... 8
   2.1. Preliminaries ......................................................................................................... 8
       2.1.1. Federated Learning ....................................................................................... 8
       2.1.2. Edge Computing ........................................................................................... 11
       2.1.3. Real-Time Object Detection ....................................................................... 13
       2.1.4. YOLO .......................................................................................................... 14
       2.1.5. NVIDIA Jetson Nano .................................................................................. 16
   2.2. Research background ............................................................................................ 17

3. Method ........................................................................................................................ 22
   3.1. Research Approach ............................................................................................... 23
   3.2. Data Collection ..................................................................................................... 24
   3.3. Methodological Framework .................................................................................. 24
       3.3.1. Model Selection ............................................................................................ 24
       3.3.2. Training Platforms ....................................................................................... 25
       3.3.3. FL Framework Selection ............................................................................. 25
   3.4. Model Evaluation .................................................................................................. 26

4. Implementation ............................................................................................................ 27
   4.1. Dataset .................................................................................................................. 27
   4.2. YOLO Models ....................................................................................................... 30
       4.2.1. YOLOv7 ....................................................................................................... 31
       4.2.2. YOLOv8 ....................................................................................................... 33
   4.3. Jetson Nano Developer Toolkit ............................................................................. 36
       4.3.1. Setup ............................................................................................................. 37
       4.3.2. Model training .............................................................................................. 38
   4.4. YOLO-FL System .................................................................................................. 41
       4.4.1. FEDn Architecture and Setup ..................................................................... 42
       4.4.2. YOLOv8 Implementation in FEDn ............................................................... 46
       4.4.3. YOLO-FEDn Deployment ......................................................................... 49

5. Results ......................................................................................................................... 50

6. Discussion ..................................................................................................................... 54
   6.1. Previous Research ................................................................................................. 54
6.2. Methods, implementation and results.................................................. 55
6.3. Ethical and societal aspects..................................................................... 57

7. Conclusion.................................................................................................. 58
7.1. Future work............................................................................................ 61

References ...................................................................................................... 63

Table of Figures
Figure 1. Architecture design of Federated Learning (Adapted from [5]) ............ 9
Figure 2. Centralized vs Decentralized Learning in FL (Adapted from [7]) ........ 10
Figure 3. YOLO versions through the years (Adapted from [29])...................... 15
Figure 4. Comparing YOLO versions in speed and size (Adapted from [36])...... 16
Figure 5. Indoor Object Detection Dataset Tree Hierarchy ............................ 28
Figure 6. Labelling dataset images by using Roboflow..................................... 29
Figure 7. Exporting custom generated dataset in YOLOv8 version (Roboflow).... 30
Figure 8. YOLOv7 Model – Confusion Matrix ............................................. 31
Figure 9. YOLOv7 Model – Train & Validate Metrics .................................... 32
Figure 10. YOLOv7 Model – Test Batch Predictions ...................................... 33
Figure 11. YOLOv8 Model M – Confusion Matrix ........................................ 34
Figure 12. YOLOv8 Model M – Train & Validate Metrics .............................. 35
Figure 13. YOLOv8 Model M – Test Batch Predictions ................................. 36
Figure 14. NVIDIA Jetson Nano 2GB Developer Kit (Adapted from [73])........ 37
Figure 15. Training YOLOv8 Model M in Jetson Nano Kit – Confusion Matrix 39
Figure 16. Training YOLOv8 Model M in Jetson Nano Kit – Metrics ............... 40
Figure 17. Training YOLOv8 Model M in Jetson Nano Kit – Predictions ......... 41
Figure 18. FEDn Network Overview (Adapted from [76]) ................................ 42
Figure 19. MNIST PyTorch tree hierarchy in FEDn FL framework ................. 43
Figure 20. Deploying FEDn into a local machine .......................................... 44
Figure 21. Attaching clients to FEDn system in a local machine ..................... 44
Figure 22. FEDn UI Dashboard ..................................................................... 45
Figure 23. Test Accuracy Metric per round in MNIST PyTorch example, FEDn 46
Figure 24. Split dataset tree structure for 3 Clients ....................................... 47
Figure 25. Local YOLOv8 implementation into FEDn with 3 clients ............... 48
Figure 26. Evaluating training precision in YOLOv8 implementation to FEDn 49
Figure 27. CPU usage while training YOLOv8 in FEDn locally ..................... 52
1. Introduction

Considering many developments occurred in the recent years with Deep Learning achieving state-of-the-art results in many domains, Machine Learning (ML) has been affected significantly and it has advanced in a great pace. Having in mind that in our daily basis, there is a vast requirement of big data to train these models and it can be even more difficult to cope with when the data we use is private or sensitive [1]. It is worth mentioning the fact that training these models in a centralized approach can be the genesis of data privacy breach and security issues.

This is why it comes to our attention that Federated Learning (FL) can be useful, where multiple clients (involved parties) can collaborate together to train the models without sharing their internal datasets. This is the trivial reason that in this thesis, Federated Learning is proposed as a solution to address the challenges of collaborative Machine Learning and data privacy problems.

Through the progress of this thesis, the main focus is the application of FL to Real-Time Object Detection, a challenging problem in Computer Vision that has broad applications in areas such as surveillance, robotics, and autonomous vehicles. The potential usage of FL for Real-Time Object Detection at the edge has been explored, where the data is processed locally and in real-time. The aim of this thesis is to build a Real-Time Object Detection model using the YOLO (You Only Look Once) family of Object Detection models and deploy it on the Jetson Nano external device. This device will serve as a sole client which will be connected to other clients in order to incorporate our Object Detection model into a Federated Learning system.

To achieve this, the possibility of utilizing one of the two FL frameworks: Flower and FEDn, has been explored. These frameworks provide the necessary infrastructure to enable FL and distribute the training process among multiple devices while ensuring data privacy and security. Saying this, a thorough investigation has taken place to determine which framework fits best to this project’s needs and picking
the adequate one. During the path, we performed performance evaluation of the models trained and measure their effectiveness in achieving Real-Time Object Detection at the edge.

1.1. Problem Definition

As previously described in the above text, FL is an approach which enables the training of numerous devices without the need of sharing their internal datasets. Saying this, it enables a promising approach for real-time object detection at the edge, where the computing power and the bandwidth of the devices are considered to be scarce resources. Consequently, we have come up with this research problem regarding this thesis:

*How can Federated Learning be utilized for Real-Time Object Detection at the edge, and what are the trade-offs between accuracy, privacy, and communication overhead?*

By analyzing this research problem, we intend to study how can Federated Learning contribute on balancing the need for accurate object detection, privacy of the data being used to train the model, and the amount of data being shared between devices. For instance, sharing a large amount of data between devices can increase the models’ accuracy and communication, but could put the privacy in risk. On the other hand, limiting the amount of data shared between devices can improve privacy, but could be a downgrade to the accuracy and communication. Saying this, we will break down the research problem into smaller subquestions:

**Research Questions:**
The research problem has been divided into the following research questions:

- How can FL be utilized for training shared model for real-time object detection at the edge?
- What are the trade-offs between accuracy, privacy, and communication overhead in this sphere?
- How can we improve the performance of FL for real-time object detection?
Aims:
- Analyzing the usage of Federated Learning for real-time object detection at the edge.
- Evaluating the trade-offs between accuracy, privacy and communication overhead in FL.
- Establishing an FL framework for real-time object detection that utilizes the means of edge devices to achieve a good balance between accuracy, privacy and communication (APC).

Objectives:
- Designing and implementing an FL framework for real-time object detection.
- Evaluating the APC factors of the proposed frameworks.
- Implementing the framework solution with an edge device (Jetson Nano developer toolkit) and evaluate its performance.

Significance:
This project aims to be relevant in a Data Science perspective, where we intend to include a step-by-step analysis of the Data Science process. We believe that this project will add value to the classification problems in object detection domain, by implementing it in a FL environment and adding elements of privacy, communication between devices and improving the accuracy by utilizing the power of an FL framework.

Considering the relation with previous studies in this area, our work contributes on FL usage in RTOB models, especially in the latest model to date, YOLOv8. This thesis utilizes the means of Federated Learning, Real-Time Object Detection models and Edge Computing, to offer an inclusive approach between these three concepts by establishing an FL framework for RTOB at the edge and observe the results of this combination in various environments.

1.2. Content outline

The overall structure of this research work consists in seven chapters, including this introductory chapter. Chapter 2 elaborates on the background on this research, focusing on preliminaries and related
work in previous research. Chapter 3 is concerned with the methodology used for this thesis. Chapter 4 describes in details the implementation that has taken place, the workflow being followed and the models trained and evaluated. Chapter 5 is concerned with the results elaboration and their analysis in a FL perspective. Chapter 6 follows discussions regarding our findings, reflections on the decisions being made and the methodology being followed. Finally, Chapter 7 summarizes the conclusion points we make into this thesis and opens the door for future developments that may follow.

2. Background

It is important to understand the existing research background in the spectrum of Federated Learning and how it affects our thesis. This chapter has been divided into two subsections; preliminaries and research background. The first subsection gives a brief presentation on the basic terminology and developments on FL for real-time object detection, edge computing and devices, NVIDIA Jetson Nano toolkits and YOLO library models. The latter subsection treats the related work in this field and analyzes their relevance to our study.

2.1. Preliminaries

2.1.1. Federated Learning

Artificial Intelligence (AI) and Machine Learning (ML) have been widely used in the recent decade for various purposes and spheres: image, text, speech classification etc. [2]. Considering their wide spread, we cannot neglect the privacy concerns traditional ML may rise by overloading the volume of central data centers [3]. The more data we store, the more difficult it is to ensure privacy, accuracy and a balanced workload in centralized systems. Saying this, the concept of Federated Learning has been brought up to the conversation, a distributed AI/ML technique able to create a global model and train it into multiple decentralized clients.
FL term has initially been mentioned by Google [1], as a proposed framework to address the issue of increasing number of devices (mobiles, IoTs, AIoTs), sensors on these devices (microphones, cameras, GPS) and the difficulty of maintaining them into centralized systems. AIoTs (Artificial Intelligence of Things) represents a combination between AI technologies and IoT (Internet of Things) infrastructure, where the main purpose is gathering large amount of data in order to create efficient IoT operations. Google used its Gboard keyboard and federated learning to train a neural language model for next word prediction [4]. The model outperformed a model created using the standard approach of collecting and training data on a central server, showing in this way improvements not only in privacy, but also in accuracy. As visualized in the following figure, the FL architecture consists in a 4-step process.

This cyclic architecture starts with transferring the gradients from the local clients to the global centralized server. It continues with updating the model in the centralized server and sending back the updated model alongside the weights to all the local models and lastly each client performs training to update their local models with their private datasets, based on the weights received by the global model. Talking about architecture, FL has two main subsections: Centralized and Decentralized architectures [6], in a sense of model aggregation and the way each node (client) communicates within the system.
The first architecture design refers to the already mentioned concepts so far: having a central server which contains the global model and other clients train their models locally based on the updated weights they retrieve from the main server (master). The second design is concerned with an all-way communication between the nodes in a FL system, where the data is still being kept locally and the weights are updated via a common algorithm used and encrypted in every transaction. The latter one has been supported by various researchers, and considering the rapid developments and its privacy success rate in cryptocurrencies, blockchain is proposed as a plausible fit to implement this design [8].

The researchers compare Federated Learning (FL) training with training on a centralized server, where data is stored on a single computing machine. While centralized learning is currently considered the most accurate approach, it does have its own drawbacks related to privacy and security. Results from several experiments indicate that FL with a centralized server shows comparable performance and reliability to the traditional central approach of training models on a single machine. A high importance should be put into the selection of FL clients, since they are vital to the overall system’s performance. Potential issues with the connection, upload/download and response time while updating the weights, their processing unification etc., can reduce the performance of the global model significantly.
Considering the operating way of a FL system, we can mention two main categories: *Horizontal FL* and *Vertical FL* [9]. The first category regards the division of the data by example, for instance having multiple clients owning their own data but sharing performance-evaluating metrics. On the other hand, vertical FL is divided by features, meaning that two or more clients may hold different chunks of data for the same topic. For instance, government institutions and private companies having different kind of data for the same people. Even though the datasets are being kept locally for each client, there exists a probability of cyber attacks to the global model and reverse engineering to the local models by using the weights as trackers. This is why data encryption and security protocols are to be taken into consideration. Various research papers have investigated into the security concerns on FL and the adequate approaches to protect it from malware intentions [10] [11]. Experimental results performed by these researchers indicate that privacy is not a built-in attribute of collaborative learning algorithms like FL, and therefore secure applications that can leak private information need to be carefully reviewed on an individual basis. In order to make the generated gradient from this reconstructed attack sample resemble the actual gradient computed on the local training data, this reconstruction attack seeks to incrementally add the generated small noises to the attack sample. By examining the shared local gradient update parameters or the weight update vector, the attackers are able to access the private local training data.

It is relevant to mention that using encrypted communication techniques surely does improve the privacy, but on the other hand, increases the processing time and requires more computational resources [12]. Several techniques have been proposed to improve this trade-off, such as homomorphic encryption for cross-silo devices [13], gradient-based aggregation, blockchain-based systems that uses filters and timestamps to verify and authenticate clients in order to save time and resources and in the same time ensure privacy [14].

2.1.2. Edge Computing

Edge computing (EC) is a distributed computing framework that allows data storage and its computation to be located closer to the edge of the network, close to where data is generated. This can increase system efficiency overall and reduce latency and bandwidth loss. As
the number of connected devices and the volume of data created continue to rise, edge computing is becoming more and more popular. This architecture brings cloud computing (CC) services closer to the data sources, lowering latency and bandwidth costs while enhancing network resilience and availability [15].

EC term has initially been brought up as MEC (Mobile Edge Computing) by ETSI (European Telecommunications Standard Institute) in 2014, defined as a new platform that provides computation capabilities to mobile subscribers [16]. Another wider term, Fog Computing, has been put up by Cisco as a more extensive version of MEC, where the concept of edge devices is expanded to include everything from smartphones to set-top boxes [17]. In order to train their models in an EC-based standard Deep Learning (DL) system, data producers must constantly communicate and share data with other third parties, edge or cloud servers [18]. Due to the strict legal requirements, privacy flaws, and high bandwidth requirements, this design is frequently unsustainable. This is why the need for a distributed system that enables edge devices to keep their data private and local, becomes more evident and seeks solution via FL.

The devices utilized for their computational power has been divided in two big categories: Cloud and Edge devices. Following the current developments of bringing the devices closer to the data sources puts an advantage to the edge devices comparing to the cloud ones. Some of the differences worth mentioning between these two paradigms include:

- **Data**, where the dataset is being stored centrally in the data centers in CC, while in the case of EC, it is stored locally in the edge devices.
- **Computational resources**, where CC has an advantage on this matter, having higher performance, availability and storage than EC.
- **Connectivity**, is a factor that has its own challenges in both domains; CC needs a stable internet connections without any timeout, while EC suffers from low internet connection and limited resources in this aspect.
- **Latency**, CC is a domain where the data processing does not need to be urgent and it can take considerable time, while EC offers real-time data processing.
Considering the above factors, the decision on choosing one paradigm over the other is based on what is the nature of the project and which is the trade-off we are willing to make. The ability to cut costs while maintaining maximum operational effectiveness is EC’s largest advantage [19]. When data processing takes place at an edge facility, there is no need to transfer it to a central service provider or provide a lot of storage space, which saves money while achieving great efficiency. Another advantage is that computations are performed at or near the edge, on or near edge devices, and are faster than those performed through cloud computing, especially in circumstances when real-time response is required [20]. On the other hand, exploiting computational resources for AI and ML processes was not practicable at the network's edge due to the restricted computing and power resources available. Demanding ML models are frequently trained using cloud computing instead, placed on strong devices and graphics cards connected to a dedicated cluster.

Alongside edge computing, come edge devices, which include every industrial/customer kind of device that are designed to build and train AI/ML models [21]. By researching the current approaches on edge devices, our goal is to investigate the possibilities of training and deploying AI models, but in the same time creating new real-time object detection models on the edge on devices such as Jetson Nano family toolkits, different mini TPUs, small PCs or laptops.

2.1.3. Real-Time Object Detection

Real-Time Object Detection (RTOB) is a Computer Vision technique that can instantly detect and categorize items in a video or image. The algorithm used must be able to process and comprehend the data quickly enough to keep up with the video or image stream, which makes this a difficult assignment. In order to achieve this technique, a variety of approaches can be pursued. One of them is Convolutional Neural Network (CNN). This approach is ideal for processing images. A large dataset of photos with the items selected according to our needs can be used to train CNNs. Once trained, the CNN may be used to detect and identify the items in new photos or videos. Deep Learning techniques have received a major increase in
the last years, following its need in multiple areas of life: CCTV, crowd detection, flow detection, fire detection [22], and many others.

A worth-mentioning fact about EC is that enables object recognition and other deep learning results by using pre-trained models to edge devices, which are located around sensors. When compared to cloud servers, the processing resources of edge devices – which are often CPUs, GPUs, or FPGAs – are restricted. As a result, numerous experiments have been carried out to perform effective deep learning inference on edge devices with limited resources. These include the development of model compression techniques like quantization and pruning of deep learning models for resource-constrained edge devices [23], the design of lightweight models with reduced weights and parameters such as YOLO family of models [24] and MobileNets [25], for use in edge computing environments.

Following the developments in data monitoring and surveillance, and the increase of focus towards privacy protection, devices in IoT and AIoT systems are more and more difficult to share, so this is where the need for Federated Learning is observed [26]. With FL, edge computing not only prevents user privacy at the edge from being compromised, but also reduces down on neural network model training time and completely employs the data in each edge client.

Regarding the scale of FL and nature of devices, it is relevant to share the two main categories of them: cross-device and cross-silo [27]. Their main difference is their association with the type of clients: the first one relates to small devices (IoTs, mobiles, etc.) and the latter one relates to organizations. Cross-device category represents a higher number of clients, with lower computational power, and having the data divided horizontally. On the other hand, cross-silo category represents fewer clients, but with more computational and storage options, supporting both horizontal and vertical FL [28].

2.1.4. YOLO

YOLO (You Only Look Once) is a series of object detection algorithms that are renowned for their high accuracy and speed, establishing in this way a reliable and accurate identification of objects in images and videos. Since its genesis, YOLO family of algorithms
has revolutionized the industry of computer vision and deep learning while applying improvements and reducing loss through all its lifetime for each iteration (see Figure 3), from YOLOv1 to YOLOv8, the latest version to date.

An overview of the YOLO family versions and its main features are summarized in the below tabular format.

<table>
<thead>
<tr>
<th>Version</th>
<th>Year</th>
<th>Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>YOLOv1 [30]</td>
<td>2015</td>
<td>Single-stage object detection; Darknet24 backbone; CNNs; Fast, Accuracy Low</td>
</tr>
<tr>
<td>YOLOv2 [24]</td>
<td>2016</td>
<td>Better backbone Darknet24; more sophisticated loss function; Accuracy improved</td>
</tr>
<tr>
<td>YOLOv3 [31]</td>
<td>2018</td>
<td>New backbone Darknet53; anchor boxes introduced; Better Accuracy</td>
</tr>
<tr>
<td>YOLOv4 [32]</td>
<td>2020</td>
<td>New backbone CSPDarknet53; hyperparameter tuning; Darknet; Very accurate</td>
</tr>
<tr>
<td>YOLOv5 [33]</td>
<td>2020</td>
<td>New backbone CSPDarknet53; data augmentation; Pytorch; State-of-the-art Accuracy</td>
</tr>
<tr>
<td>YOLOv6 [34]</td>
<td>2022</td>
<td>New backbone EfficientRep; quantization scheme; new regression/classification losses</td>
</tr>
<tr>
<td>YOLOv7 [35]</td>
<td>2022</td>
<td>Improved backbone v7Backbone; re-parameterized convolution; batch normalization</td>
</tr>
<tr>
<td>YOLOv8 [36]</td>
<td>2023</td>
<td>Improved backbone CSPDarknet53; larger batch size; most accurate version</td>
</tr>
</tbody>
</table>

Following the displayed tabular comparison between models, we have decided to pursue the implementation of this project using
YOLOv7 and YOLOv8 while training them via a custom dataset for indoor object detection, and picking the most optimal one. By observing the documentation in the official Ultralytics website and their GitHub corresponding repository about YOLOv8, we can understand that this model is way faster than its predecessors by using less parameters (see Figure 4).

![Figure 4. Comparing YOLO versions in speed and size (Adapted from [36])]({})

2.1.5. NVIDIA Jetson Nano

Jetson Nano is a developer toolkit built by NVIDIA in order to implement IoT solutions by utilizing the power of GPU computation [37]. Machine learning and deep learning applications can be run on the Jetson Nano platform. Although the GPU-powered platform may be utilized to set up online learning models and train models, it is best suited for using pre-trained AI models for high-performance inference in real-time [38]. AI is not the only domain where Jetson Nano GPU capabilities can be used on, though. Other disciplines like gaming, robotics, autonomous vehicles, HPC and creative design are very well-suited for this toolkit by NVIDIA.

Considering the above-mentioned details, with this project we aim to apply a horizontal FL system by utilizing one of the most recent YOLO models so far (v7 or v8) on a cross-device Jetson Nano 2GB toolkit to train an RTOB model and integrating it into this system.
2.2. Research background

As previously discussed in the preliminaries section, Real-Time Object Detection (RTOB) is a highly challenging AI task which requires vast computing resources and high speed. Its combination with Edge Computing (EC) ensures bringing this computation closer to the edge of the network, consequently improving overall performance. Federated Learning (FL), on the other hand, promises to be a supportive discipline for RTOB in Edge Computing, by increasing collaboration and efficiency among the devices.

This literature review will examine recent developments in Federated Learning applications within the AI/ML domains, underlining the difficulties presented by non-identically distributed data. We will highlight popular FL algorithms and present researchers' perspectives on their implementation. Additionally, we will introduce available open-source FL frameworks.

Implementing FL algorithms has received a lot of attention recently in order to support efficient Machine Learning models. Researchers have put a lot of effort into analyzing the role of FL into supporting privacy-oriented approaches in many ML domains, including deep neural networks. We can mention the case of Google, which suggests a scalable production solution that supports deep neural network training on tens of millions of devices [39]. FL has been used more and more as an enabling technology for mobile edge network optimization in addition to being an enabling technology for ML model training at those networks. Traditional network optimization techniques based on static models perform very poorly in modelling dynamic networks due to the computing and storage constraints of increasingly complex mobile edge networks. Saying that, in order to optimize resource allocation, a data-driven Deep Learning (DL) based strategy is becoming more and more common [40]. Considering the developments in DL and Deep Neural Networks (DNN), it is safe to consider FL's potential to be an enabling technology for improving mobile edge networks, such as cell association, compute offloading, and vehicle networks [41]. Speaking of implementations, FL has been utilized with many AI/ML tasks, such as: images classification, Reinforcement Learning (RL) [42], text generation [43], transfer learning [44] and algorithms like: Gradient Boosting trees [45], Long Short-Term Memory (LSTM) networks [46], etc.
Considering the nature of the data, there have been various studies on the categories and how they affect FL systems. The two main categories are Independent and Identically Distributed (IID) data and non-IID data, homogeneous vs heterogeneous data [1]. As their definitions suggest, the first category regards equally distributed data with common similarities. The latter one regards data being notably different from clients in the edges of the network. We must note though, that having non-IID data and imbalanced data are quite different terms. While non-IID focuses more on the nature of the datasets and the similarities between them, the imbalance problem means different clients having different volume of the data; some may have small datasets, other may have larger ones.

Speaking of non-IID data, many researchers have proposed their solution to this issue. Although Federated Averaging (FedAvg) [47] algorithm is considered to be one of the most efficient ones in training FL models, it is also known that it does not excel in every case it faces non-IID data, decreasing in this way the performance of global model while working with this algorithm. Although FedAvg has its own restrictions in this matter, many researchers have proposed frameworks that addresses non-IID data, based on FedAvg. One of them is the research done by Zhao et al. [48], where they propose a shared data subset being distributed globally in order to enhance the training using this algorithm. Similarly using FedAvg algorithm, Li et al. [49] emphasize a comprehensive convergence analysis of this algorithm when training AI models with non-IID data.

Wang et al. [50] study a general approach of dealing with heterogeneous inconsistencies in federated systems. They offer a rational explanation for the global model's slow convergence and alignment distortion caused by data consistency. The FedNova algorithm, which they have proposed as part of this research, focuses on a normalized averaging technique that gets rid of object consistency while maintaining fast error convergence. In federated networks, this technique aims to addresses problems with heterogeneity. The re-parametrization that exists in FedAvg was attempted to be generalized by the authors. This algorithm's preliminary tests demonstrate that it is much more accurate and stable than FedAvg.

In order to provide faster convergence when dealing with AI models' training and address the issue of non-IID data, Karimireddy et al. [51] have established a well-known algorithm: Stochastic Controlled
Averaging For Federated Learning (SCAFFOLD). In their local updates, the authors put a lot of effort towards minimizing the data collection variance. They use experiments to show that SCAFFOLD needs a lot fewer communication rounds and is unaffected by client sampling or data heterogeneity.

Another approach on data heterogeneity and imbalance is concluded to be multi-task learning [52] by many researchers. The FEDPER approach [53] is build based on the concepts of multi-task learning by Arivazhagan et al. Participants in FL share a set of base layers that have been FedAvg-trained under this framework. Then, using their own local data, each participant trains a new set of personalization layers independently. Given the wide range of participant preferences, this approach is particularly useful for developing recommender systems. Another framework from Bezdek and Hathaway, MOCHA [54], is also built based on multi-task concepts. MOCHA uses an alternating optimization approach to solve the issues with minimization and heterogeneity.

In order to implement a FL system, we reviewed many open-source FL frameworks, including the following ones [55] [56]:

1) **TensorFlow Federated (TFF)** by Google [57], is an open-source FL framework whose main motivation was incorporating mobile keyboard predictions and on-device search. It contains two API layers: Federated Core (FC) API and Federated Learning (FL) API. FC API is a programming environment that allows developers to write distributed computations that can run on a variety of devices and platforms. FL API, which is built on top of FC API, is a high-level API that makes it easy to use federated learning with existing machine learning models.

2) **PySyft** by OpenMined [58], is a Python library that implements FL for research purposes, encrypted computational tasks and differential privacy. It offers two types of computations: Dynamic computations over unreachable data and Static ones that can be carried out later in a different computing environment.

3) **Federated AI Technology Enabler (FATE)** by WebBank’s AI department [59], is an open-source project whose aim is to
provide a secured federated AI ecosystem by supporting cluster and standalone deployment setups. It implements a series of secure computational protocols to support big data collaboration with data regulation policies. FATE builds secure federated systems by using various algorithms such as deep learning, decision trees, etc.

4) Paddle Federated Learning (PFL) by Baidu [60], is an FL framework with an Apache license, which uses a deep learning (DL) platform to process partitioned data horizontally and vertically. PFL uses Neural Networks (NN) and Logistic Regression (LR) models and it also applies Differentially Private Stochastic Gradient Descent (DPSGD) to implement FL algorithms like FedAvg and SecAgg.

5) LEAF by Carnegie Mellon University [61], which has applications for FL, multi-task learning, meta-learning, and on-device learning, is a benchmarking system for learning in federated contexts. Many benchmark datasets like MNIST, Twitter sentiment140, Reddit Language Modeling have been partitioned in a FL context and this new implementation of existing datasets provides trustworthy research comparisons.

6) Tier-based Federated Learning (TiFL) [62], is a prototype system built in a FL environment following Google's FL architecture, which classifies customers into tiers depending on how well they performed during training and chooses clients from the same tier in each training cycle. This approach helps reducing the straggler issue being brought by non-IID data heterogeneity and data quantity.

7) FedML [63], is a free research source and benchmark that makes it easier to create new FL algorithms and conduct objective performance comparisons. For users to conduct experiments in various system contexts, FedML provides three computing paradigms (distributed training, mobile on-device training, and standalone simulation). Offering solutions in multiple domains such as financial industries, autonomous vehicles, healthcare, Web3 and Blockchain, FedML provides access in multiple features via their generic API design.

8) OpenFL by Intel [64], is an open-source project whose main aim is to inject FL on sensitive data. Although OpenFL uses certificates to secure communication and provides deployment scripts in bash, much of this must be handled by the
framework’s user. OpenFL is made up of two parts: the colaborator, which trains global models using a local dataset, and the aggregator, which gathers model updates and combines them to produce the global model.

9) *Substra* by Owkin [65], is an FL software framework which is mostly focused on medical applications with the purpose of data privacy and ownership. Supporting a variety of interface means such as CLI for admins, Python library for data scientists, UI for project managers and other users, it generally provides key features as security, privacy and traceability.

10) *Flower* [66], is an open-source framework that enables training, analytics and evaluation among highly heterogeneous FL device scenarios. Flower provides the infrastructure to move, evaluate and aggregate ML models regardless the workload, framework or programming language. Its foundation is built using DL frameworks like PyTorch, Tensorflow, etc.

11) *FEDn* [28], is an open-source framework which enables distributed, scalable and resilient implementation of a FL system that offers support for both cross-silo and cross-device scale scenarios. By using MapReduce programming pattern as an architecture foundation, FEDn system consists of three logical tiers:

- *Clients*, local entities responsible for their datasets and locally updating their models.
- *Combiners and Reducer*; The first ones make sure that updates are being coordinated accordingly to their assigned subset of clients and alongside with combiners make up the FEDn network.
- *Controller and Discovery service*; The first one is in charge of managing global state, keeping track of global models, and coordinating the entire computation done during a global training round. On the other hand, receiving client connection requests and allocating clients to network combiners are the responsibilities of the discovery service component.

FEDn uses gRPC, a high-performant, open-source remote procedure call (RPC) framework developed by Google, to communicate between layers and entities in its system. While combining gRPC with secure protocols as Transport Layer Security (TLS) and Secure Sockets Layer (SSL), FEDn
makes sure that communications are encrypted, protecting data in transit from eavesdropping and manipulation.

12) **Clara and NVFlare** frameworks by NVIDIA, are open-source projects to support their toolkit devices into FL systems. NVIDIA Clara [67] is a platform whose purpose is providing solutions for healthcare developments, being a collaborative platform for healthcare developers, researchers, and manufacturers of medical devices who are developing AI solutions to enhance healthcare delivery and quicken medication development. AI advancements are being made possible by Clara's domain-specific tools, pre-trained AI models, and expedited applications across a variety of industries, including genomics, NLP, imaging, medical devices, drug development, and smart hospitals. On the other hand, NVFlare [68] is a Python library which serves as an extensible SDK and allows researchers to train and adapt various ML models and libraries like Tensorflow, PyTorch, XGBoost, Scikit-learn and so forth into a FL paradigm. By also supporting FL algorithms such as FedAvg, SCAFFOLD, FedProx, etc., it allows for moving FL workloads from research state to real-world production deployment.

By analyzing some of the most relevant open-source FL frameworks, we aim to evaluate and comprehend their structure and utilize the one that fits the best to our RTOB problem. Considering the options, we plan to dig deeper into two of the above frameworks, FEDn & Flower.

3. Method

In this chapter, we will outline the methodological approach being followed in this project for the purpose of investigating the feasibility and efficacy of RTOB using FL in Edge Computing environments. The chapter includes an outline of the research approach, the data collection methods, the methodology for model training and evaluation and most importantly, the reasoning behind these choices, all in the context of addressing the above-defined research problem.
3.1. Research Approach

The research challenge stated in the introduction, "How can Federated Learning be utilized for Real-Time Object Detection at the edge, and what are the trade-offs between accuracy, privacy, and communication overhead?" is what our study tries to resolve. The research strategy adopts a complete approach that combines theoretical understanding, contextual awareness, and ethical considerations to shed light on the numerous opportunities and challenges being present in the research problem.

Considering the theoretical alignment of our study, we focus on the related work, established frameworks and the ongoing developments on data privacy, accuracy and communication factors in regard to Federated Learning systems. Being able to jointly comprehend multiple FL frameworks and the work done in this direction, motivates our research to detect pros and cons and choose the adequate approach to implement our solution in. Saying this, our decision on focusing our research in related work contributions, was clear from the beginning. A user involvement approach does not make much sense in our study, consequently, qualitative insights are gathered through an in-depth exploration of the broader context surrounding the research problem, rather than user surveys or interviews. Regardless the absence of end users in our research, our methodology remains qualitative, including relevant literature observation, multiple technical use case scenarios, open-source projects’ community experts, who provide valuable perspectives on the challenges, advancements and opportunities in deploying Federated Learning for RTOB at the edge.

Regarding societal and ethical considerations, an important focus has been put on this regard, even though direct user feedback might not be part of this study. The ethical concerns of using federated learning for RTOB, particularly in edge computing situations, are investigated using qualitative methodologies. We guarantee a thorough examination that goes beyond technical performance by analyzing privacy issues, regulatory frameworks, and potential impacts on society.
3.2. Data Collection

Aligning with our research problem, a suitable dataset for Real-Time Object Detection has been chosen. We have decided to focus our research into Indoor Object Detection and this is the sphere we want to build our custom YOLO model and deploy it to the adequate FL framework. Indoor objects came to our interest for the trivial fact that can be utilized for many purposes. We can mention some of them: Smart Home systems, Retail and Shopping, Healthcare/Home Healthcare, Education and Research. Regarding the healthcare perspective, indoor object detection algorithms can help monitoring patient movements, medical equipment availability, assisting elderly patient in their households by detecting their surrounding environments and detecting falls or unexpected behaviors. More details regarding the chosen dataset and the data preparation will be provided in the Implementation chapter.

3.3. Methodological Framework

The methodological approach used in this study's investigation of Federated Learning for Real-Time Object Detection at the edge is described in this section, along with a comprehensive analysis of the trade-offs between accuracy, privacy, and communication overhead. Model selection, training platforms, and careful examination of open-source federated learning frameworks are all part of the study approach's comprehensive research.

3.3.1. Model Selection

Our approach on selecting YOLO models for our object detection problem comes naturally and strategically. YOLO family of object detection models (especially YOLOv7 and YOLOv8) are well-known for their capabilities on balancing accuracy and speed, two high-relevant factors in real-time applications in edge environments. This is why our focus has been on training these specific models (and different variations of them) in order to pick the most optimal one according to our needs.
3.3.2. Training Platforms

Since our research problem’s main focus is detecting the trade-offs between privacy, accuracy and communication, we decided to test our models in different environments in order to measure their performance in different conditions. We trained our YOLO models in many platforms, starting from locally (in an Ubuntu laptop machine), in Jetson Nano Kit, while implementing YOLO model to the FL framework locally and simulating an FL system with local clients, and lastly in a real Cloud FL environment with real clients. Having these much options in our availability, enables us to represent the distinct considerations each platform contains regarding accuracy, privacy and communication.

3.3.3. FL Framework Selection

We took into consideration a number of open-source frameworks, including Flower and FEDn, to build the federated learning component of our study. Both frameworks have distinctive qualities that fit our study's goals. Generally speaking, FEDn places a major emphasis on privacy preservation and scalability whereas Flower emphasizes ease of integration. Flower is a framework that offers a lot in terms of compatibility, supporting most of the ML algorithms and libraries. It also offers a range of management and evaluation tools for federated learning systems, including a central dashboard for monitoring training progress and a collection of metrics for assessing model performance. FEDn, in the other hand is a way lighter solution than Flower, being specifically designed for supporting edge devices (clients). A major benefit of using FEDn is training ML models on the edge, without the need of sharing/uploading their data to the Cloud. In terms of privacy, FEDn uses gRPC as a mean of communication between clients and combiners, also combined with SSL/TLS protocols which add an extra layer of security. Considering the above factors, we opted to utilize FEDn FL framework. Several aspects directly related to the research problem influenced the choice. Our objective of ensuring data privacy in edge contexts is consistent with FEDn's privacy-preserving techniques. Additionally, our goal to balance accuracy and communication overhead
was in line with FEDn's capacity to effectively handle various federated learning scenarios. Although Flower has beneficial features, FEDn comes more in handy to our implementation and research problem.

3.4. Model Evaluation

In order to evaluate the performance of the object detection models in different environments, we have used some of the most used classification/detection metrics.

- **Confusion matrix**, a tabular representation of an algorithm’s performance. It displays the number of true/false positives (TP, FP) and true/false negatives (TN, FN) for each class.
- **Precision**, the percentage of the objects being correctly identified as belonging to a certain class.
- **Recall**, the percentage of objects being part of a certain class, that are actually identified as such.
- **mAP (Mean Average Precision)**, probably the most used performance evaluation metric in object detection model, it represents the average of Average Precision metric across all the classes in a certain model. This metric has two main subsections; mAP(5) and mAP(5-95). The "(5)" means that it is calculated at an Intersection-over-Union (IoU) threshold of 0.5 (or 50%). The "(5-95)" means that it is calculated across a range of IoU thresholds, from 0.5 to 0.95 with a step size of 0.05.

These metrics have been used to evaluate the overall performance of our models, but we also have observed the training time, CPU/GPU usage and adaptability to a certain environment/platform. The above metrics regard the “A” factor (Accuracy) in our APC problem.

In order to assess the two other factors “P & C” (Privacy and Communication), we will evaluate the tools that our chosen FL framework (FEDn) offers, by monitoring how clients coordinate with each other in terms of privacy and communication in different environments.

We have outlined a detailed roadmap in this methodological research to investigate the feasibility and efficacy of RTOB using Federated Learning in Edge Computing environments. Theoretical understanding, qualitative insights, model selection, and platform evaluation are
all parts of our method, which has been carefully developed, having
in mind our research problem and the study’s objectives and aims.
We have built a strong basis for addressing the complex interplay be-
tween accuracy, privacy, and communication overhead in our pursuit
of an integrated approach by embracing a strategic focus on privacy,
scalability, and real-world application.

4. Implementation

Having evaluated many object detection models and relevant related
work, we have chosen to narrow our focus in a specific area of object
detection, the one in indoor objects. We will discuss the dataset we
have used in this chapter, by describing the specifics, classes, structure
and why it is relevant for our case. Another topic of subject will be the
comparison between YOLO models (YOLOv7 and YOLOv8) in gen-
eral, how they fit in our project, and deciding which one is the proper
model to move forward.

We will also elaborate on setting up the edge device we have used in
our project, Jetson Nano Developer Toolkit. Its configuration, setup,
training, supporting technologies and training process will be dis-
cussed. Another relevant subject in our work has been choosing the
most adequate open-source FL framework to implement our RTOB
solution in. Flower and FEDn have been the frameworks we have been
observed, and as described in the above chapter, we have decided to
move on with FEDn framework. Its architecture, system designation
and configuration will be discussed. Lastly, but not less importantly,
we will describe the implementation of our real-time object detection
YOLO model in the FEDn system and also details regarding the de-
ployment and training in a production environment, where clients
(such as Jetson Nano) can train simultaneously using this system.

4.1. Dataset

As mentioned above, we have focused our research into Indoor Ob-
ject Detection and we have decided to work with an open-source da-
set in Kaggle that has data for indoor objects, the purpose of which
is to detect indoor obstacles for blind people [69]. The dataset con-
tains 10 classes: Door, Opened Door, Cabinet, Cabinet Door, Refrigerator Door, Window, Chair, Table, Sofa/Couch, Pole. The download size of this dataset is 385 MB and it has a tree hierarchy of this nature:

![Figure 5. Indoor Object Detection Dataset Tree Hierarchy](image)

It holds around 1000 training images, 200 validation images and 100 test images, with each of them having their corresponding label inside the labels directory. In the main root of the dataset, we also have the configuration YAML file, which specifies the number of classes, their technical names, and the path for each of the above directories: the dataset itself and train/valid/test subdirectories.

Even though the chosen dataset had its own training, validation and testing images and labels, we have decided to utilize an online framework called Roboflow [70]. Roboflow is a framework for computer vision which makes it simple to create and use computer vision models. It offers resources for data augmentation, implementation, training, and labelling. Developers, researchers, and organizations use Roboflow to create a range of computer vision applications, such as semantic segmentation, object identification, and image classification. Roboflow allows you to upload your custom dataset and improve its labelling, fix missing data and also generate new datasets based on the provided ones.
And this is the main reason why we felt the need to use this service. By facing multiple challenges like: missing labels, CPU memory throttling and many environments that we need to train our YOLOv8 model (local machine, Jetson Nano Kit, FEDn FL framework on Cloud, we could use the features of Roboflow to create suitable versions of our datasets for each environment.

The below image represents custom labelling in a malformed file, which lacked the assigned label from the original dataset.

![Image of custom labelling in a malformed file](image.png)

*Figure 6. Labelling dataset images by using Roboflow*

Additionally, Roboflow enables generating a new version of the dataset by customizing characteristics like:

- **Source images**: In this section, we can upload more images to our dataset.
- **Train/Test/Valid Split**: In this section we can rebalance the distribution between the sets accordingly to our needs.
- **Preprocessing**: We can apply auto orientation and resize in our images in order to achieve a better training time and improve performance. Other options available (Grayscale, Tile, Auto-adjust, etc.).
- **Augmentation**: Probabilistically applying Grayscale/Flip/Rotate/Hue etc. to a customized subset of training images.
- **Generate**: The last step is concerned with the dataset size we want to export.

Finally, we can export our custom generated dataset in the format we need, which in our case is YOLOv8 model, but we can see that we
have plenty of other models, which make this tool easily compatible with the main state-of-the-art object detection models and not only.

![Figure 7. Exporting custom generated dataset in YOLOv8 version (Roboflow)](image)

4.2. YOLO Models

When it comes to choosing a model for object detection, it is difficult not to mention the YOLO models. As mentioned in the Background chapter, YOLO (You Only Look Once) is a series of open-source object detection algorithms, based on end-to-end Deep Neural Networks. The superiority of these models is based on their approach of predicting on bonding boxes and class probabilities in the same time, all at once (as the name suggests). These models can support various AI tasks, such as: Object detection, Instance Segmentation, Pose/Keypoints and Classification. The key factor of this series of object detection models being so accurate and qualitative, has to be the fact that is an open-source project and numerous collaborators have worked in different versions of YOLO through the years. Having listed the main specifics of all YOLO models in the previous chapters, we will focus our implementation into the last two models of this family: YOLOv7 & YOLOv8. Training these models and observing their resulting metrics will be key indicators on what we will continue to do next.
4.2.1. YOLOv7

YOLOv7 is a state-of-the-art RTOB model released in July 2022 by Wang et al. [35] and in that time it surpassed all the known object detector models in terms of accuracy and speed. By utilizing strategies like extended and compound scaling, bag-of-freebies, and multi-scale training, YOLOv7 achieves high performance. On numerous datasets, including the Microsoft COCO dataset [71], it has been demonstrated to outperform other real-time object detectors. Microsoft COCO is a large-scale dataset used for various ML tasks, such as object detection, segmentation, and so on. It consists of over 300,000 images, annotated in 80 classes. In addition, at a rate of up to 160 frames per second, YOLOv7 can detect up to 100 items in an image.

We decided to train YOLOv7 model in our custom indoor detection dataset, setting some of the main parameters to: 20/50/100/200 Epochs, 4/16/32 Batch Size, 416x416 Image Size and Initial Learning Rate 0.01.

The above figure shows the confusion matrix and the differences between True/False Positives (TP/FP) and True/False Negatives (TN/FN). We can observe that classes like Door, Cabinet Door, Refrigerator Door and Chair have TPs higher than 50%, meaning that in most of the cases they were predicted correctly. On the other hand, classes like Window, Table, Opened Door and Pole have higher than 50% FNs for background, meaning that these instances were mistakenly predicted as background. The last two classes stand out, having
100 % FN value. This is due to the lack of labels in training for these classes, being considerably low related to other classes.

The above image shows the Box, Objectness, Classification, Precision, Recall and mAP results for training and validate processes in YOLOv7. While Box and Objectness show how efficiently the bounding boxes were created to detect an object inside an image or video, the classification shows how the result of detecting the objects in these object. From the given plots, we can see that Precision, Recall, mAP has an increasing trend, which gives us the perception that with more epochs (iterations), we could have got better results. Objectness, on the other hand, seems to have a non-properly distribution around the epochs, meaning that we cannot presume its results will get better if adding more epochs.

Here we will display our main evaluation metrics for YOLOv7:
- Precision is 0.768
- Recall is 0.3729
- mAP[.5] is 0.416
- mAP[.5:.95] is 0.2414

The following image displays some predictions made in test images after the training of YOLOv7 model.
4.2.2. YOLOv8

YOLOv8 model, in the other hand, is the latest version of YOLO models, being built this year (2023) by Ultralytics [36]. It is the fastest and most accurate object detection algorithm to this date. It contains a set of distinguished features like:
- CSPDarkNet53, which is a brand-new backbone network that replaces DarkNet53 and is more precise and efficient than YOLOv5.
- PANet, which enhances object detection precision by more effectively combining features from various scales.
- GIoU loss function, which in comparison to the IoU loss function used in earlier YOLO models, is more resistant to object interference and displacement.

We have decided to train locally multiple versions of YOLOv8 models, like Model M, N and X [72]. These versions of YOLOv8 represent pre-trained models, containing different training details:
- **Model M**, trained using COCO dataset with image size of 640x640, mAP (5-95) of 50.2 %, CPU speed of 234.7 ms and using 25.9 million parameters.
- **Model N**, trained using COCO dataset with image size of 640x640, mAP (5-95) of 37.3 %, CPU speed of 80.4 ms and using 3.2 million parameters.
- Model X, trained using COCO dataset with image size of 640x640, mAP (5-95) of 53.9 %, CPU speed of 479.1 ms and using 68.2 million parameters.

We have performed tests regarding the above three YOLOv8 models in our custom indoor detection dataset, setting some of the main parameters to: 20/50/100/200 Epochs, 4/16/32 Batch Size, 416x416 Image Size and Initial Learning Rate 0.01. Below we will show evaluation metrics regarding YOLOv8 – Model M, the highest performing algorithm among M, N and X models.

The above figure shows the confusion matrix and the differences between True/False Positives (TP/FP) and True/False Negatives (TN/FN). Similarly to YOLOv7, we can observe that classes like Door, Cabinet Door, Refrigerator Door and Chair have TPs higher than 50%, meaning that in most of the cases they were predicted correctly. On the other hand, classes like Window, Chair, Table, Cabinet and Pole have higher than 50% FNs for background, meaning that these instances were mistakenly predicted as background. The last two classes stand out, having 100% FN value. In comparison with YOLOv7, we can see an improvement in the Opened Door class on background FN, from 100% to 46%.
The above image shows the Box/Classification/Distribution Loss, Precision, Recall and mAP results for training and validate processes in YOLOv8. From the given plots, we can see that Precision and Recall have non-stable trends, showing high variation between epochs, meanwhile mAP shows a slow growth towards the epochs. Early Stopping techniques would have been a plausible approach to detect the most performing epoch. Here we will display our main evaluation metrics for YOLOv8:
- Precision is 0.698
- Recall is 0.62
- mAP[.5] is 0.486
- mAP[.5:.95] is 0.309.

Given the above metrics results, we can see that YOLOv8 model M outperforms YOLOv7 and we will proceed our project based on the latest, fastest and most accurate YOLO model, YOLOv8.

The following image displays some predictions made in test images after the training of YOLOv8 model M in local environment.
4.3. Jetson Nano Developer Toolkit

NVIDIA introduced the Jetson Nano Developer Toolkit 2GB, a lightweight powerful computer, in February 2021 [37]. It is intended for creators, learners, and developers that wish to create robots and applications with AI. The Nano 2GB is equipped with an ARM Cortex-A57 quad-core processor running at 1.4 GHz, a 128-core NVIDIA Maxwell GPU, 2GB of LPDDR4 RAM, and 16GB of eMMC storage. It is supported by the NVIDIA JetPack SDK, which contains a full set of tools and libraries for developing AI applications, and it runs Ubuntu 18.04 LTS. Our decision on choosing this toolkit comes naturally, considering its lightweight design and the ability to a variety of applications, such as NLP, Speech Recognition, Robotics, Edge Computing, Image Classification and most importantly Object Detection, which is the focus of our project.
4.3.1. Setup

We have setup this toolkit by following the guide on getting started with Jetson Nano 2GB Developer Kit in the official NVIDIA website [73]. Let’s have a look at the device’s hardware specifics:

Regardless its size, the Jetson Nano Kit provides most of the hardware ports a normal computer does. It contains multiple USB ports to connect input devices such as keyboard, mouse, etc. It provides internet connectivity via Ethernet or WiFi. It also contains HDMI output port to be connected with a screen, camera connector and USB-C power input port. Setup process consists in these steps:

1. **Writing the Jetson Nano image to a microSD Card**
   
   In this step we downloaded the official Jetson Nano image (NVIDIA JetPack 4.6.1), used SD Memory Card Formatter for Windows application to format our 128 GB microSD card and lastly Etcher application to write our JetPack image into the microSD card.

2. **Setup and First Boot**
   
   After having our microSD card flashed, we proceed on setting up Jetson Nano to work. There are two methods to interact with the developer kit: either "headless mode" with a link
from another computer, or with a monitor, keyboard, and mouse attached. We chose to move on with the second option, in order to have a higher GUI accessibility and interaction, considering that the first option could only be accessed via command line (PuTTY in Windows, or Terminal in Linux). We proceeded by inserting the microSD card into its slot in the kit, connected with a USB-C power supply and linked it with a keyboard, mouse, monitor.

The booting process is a normal one that Linux operation systems follow, by setting the system language, time zone, username, password and so on. An important worth-mentioning factor is selecting the APP partition size, which we did select the maximum available (MAXN) with 4 CPU cores. We also made sure to allocate 4GB disk SWAP file, in order to enhance device usage in AI models.

4.3.2. Model training

Being a toolkit that has ended its life cycle, the Jetson Nano 2GB Developer Kit faces several issues with upgraded libraries. Still, this does not mean that there is no support for this device, but the libraries being used are less up-to-date than other Jetson Nano toolkits like: Jetson AGX/NX Orin Series Kits, Jetson AGX/NX Xavier Series Kits, Jetson TX2 series, etc.

YOLOv8 model is built by using PyTorch library in Python and it requires 3.8 or higher versions of Python. Jetson Nano Kit, having the Ubuntu 18.04 OS, limits the Python version up to 3.6.

Before moving on with the YOLOv8 model training locally in the kit system, we needed to prepare the environment to do so.

We were able to upgrade to newer versions of Python by using a custom solution offered by Jetson Hacks GitHub repository “build_python” [74], where we could upgrade the Python version 3.8 – 3.11. Even though a decent workaround, it took nearly 16 hours to fully build the wheel system for the new Python version.

We decided to proceed with Python version 3.8, considering the fact that it supports YOLOv8 and also is one of the versions supported from FEDn FL framework, which we will discuss next.

We noted that even though the installation of YOLOv8 was successful, the installed PyTorch and TorchVision default versions coming with YOLO, did not have CUDA support. CUDA is a computing
platform built also by NVIDIA for the intention of utilizing GPU processing. Saying this, without CUDA support, we would not be able to unlock the potential of Jetson and we would be only working on its minimal 2GB CPU.

We were able to install custom PyTorch and TorchVision libraries, which had CUDA support, by following the steps of Hiroyuki Obinata’s i7y blog tutorial [75] on how to build PyTorch on Jetson Nano for Python versions equals to or higher than 3.7.

With the above being established, we continued training the YOLOv8 model in our custom indoor detection dataset. Having faced difficulties with voltage throttling while using GPU in Jetson Nano, we needed to deprecate some parameters in order not to cause memory failure in the system. We have performed training regarding the YOLOv8 M model (chosen as the most performing one in previous steps), setting some of the main parameters to: 10 Epochs, 3 Batch Size, 300x300 Image Size and Initial Learning Rate 0.01.

The above figure shows the confusion matrix and the differences between True/False Positives (TP/FP) and True/False Negatives (TN/FN) while training YOLOv8 model in Jetson Nano Kit. We can observe that classes only Cabinet Door has TPs higher than 50%, meaning that in most of the cases this class is predicted correctly. On the other hand, only Cabinet Door and Refrigerator Door have less
than 50% FNs for background, meaning that only these instances were not mistakenly predicted as background. This means, that the results from confusion matrix are way worse than YOLOv7 & YOLOv8 trained locally.

The above image shows the Box/Classification/Distribution Loss, Precision, Recall and mAP results for training and validate processes in YOLOv8. From the given plots, we can see that Precision and Recall have non-stable trends, showing high variation between epochs, meanwhile mAP shows a non-stable growth towards the epochs, meaning that with more epochs we could possibly get better results. Here we will display our main evaluation metrics for YOLOv8 M in Jetson Nano 2GB Developer Kit:
- Precision is 0.265
- Recall is 0.202
- mAP[.5] is 0.124
- mAP[.5:.95] is 0.066

These are decent metrics considering the deprecation in parameters we needed to apply, but we could still have had a better result with proper GPU utilization.

The following image displays some predictions made in test images after the training of YOLOv8 model M in Jetson Nano Kit.
4.4. YOLO-FL System

In this section, we focus on the foundation of our implementation, integrating the selected FEDn framework with the chosen YOLOv8 model to produce a reliable and effective real-time object detection system for edge environments. Our decision on selecting YOLOv8 comes after evaluating the metrics in the above chapter (4.3.). A more detailed analysis on the results will be displayed in the upcoming Results chapter (5.). The combination of YOLOv8's speed and accuracy with FEDn's privacy-preserving architecture results in an effective solution that tackles on the difficulties of real-time object detection while protecting sensitive information. The FEDn architecture and setup, the integration of YOLOv8 into the FEDn framework, and the consequent deployment of this system in a cloud environment are all covered in detail in this chapter's analysis of the YOLO-FL system. A comprehensive knowledge of the capabilities and contributions of the YOLO-FL system to the field of real-time object identification at the edge is reached by combining the insights provided by each subchapter into the design, implementation method, and results obtained.
4.4.1. FEDn Architecture and Setup

As briefly mentioned in the Background chapter, FEDn proposes a horizontal FL architecture, constructed by three main components: the Reducer, the Combiner/s and Clients [28]. As illustrated below, we can see that the system’s network is built by having the reducer as controller/coordinator of the training over combiners’ network. Combiner/s, on the other hand, serves as the framework for the FedML orchestration mechanism and it can be horizontally scaled in order to accommodate and expand the number of customers being connected.

![Figure 18. FEDn Network Overview (Adapted from [76])](image)

In a task-oriented perspective, Clients are data points that store private data, communicate with Combiners in order to get model updates and validations, and run code provided by the Reducer with only configuration needed before connecting. Combiners have the role of aggregating and orchestrating the model weights update from numerous clients during each training round, being executed in independent gRPC serves and evaluate the network’s capacity for connecting clients. The Reducer holds the main role of specifying the global training strategy to each combiner, model aggregation and also plays a facilitator role in connecting clients to their dedicated combiner via REST API requests. The architectural scheme of FEDn utilizes FedAvg algorithm in both combiner and reducer level, applying in this way, hierarchical FedAvg.
FEDn is a very light platform with clear instructions on how to set up your first Federated Learning system on their online documentation [76]. It includes two comprehensive approaches on how to setup the well-known MNIST dataset to perform hand-written text recognition by using two powerful libraries as Keras and PyTorch. The main prerequisites of FEDn are:

- A Python version from 3.8 to 3.10.
- Docker
- Docker Compose

Docker and Docker Compose are easily installed platforms which help on building, maintaining and deploying application at scale. And also, the Python version matches with the ones we have used locally and in the Jetson Nano Kit (3.9). FEDn also uses Mongo database in order to store the information regarding the clients, combiners, reducer, rounds, configurations and more. For object and file storage, FEDn uses MinIO, a Cloud service similar to Amazon S3 buckets used for BLOB storage and not only.

The GitHub public repository, FEDn [77], contains a step-by-step process of setting up the project in a local machine, and sufficient explanations on how to run the PyTorch and Keras examples.

The process of running this example consists on simple steps:

i. Cloning FEDn repository and locating the terminal to the “examples/mnist-pytorch” directory.
ii. Preparing the Python virtual environment to install all the necessary requirements into it (`init_venv.sh` script).

iii. Getting the dataset using PyTorch script and executing a split script to create N number of datasets for N clients we want to set regarding this experiment (`get_data` and `split_data` scripts).

iv. Generating the compute package and the seed model. The `compute package` (package.tgz) is a zipped file which contains the responsible Python file for training, validating, saving and loading models in a client environment. The `seed model` (seed.npz) is the initial version of the model which will be uploaded to the FEDn UI and be utilized for training the PyTorch-based model (`build.sh` script).

v. Deploying the FEDn system by starting the docker containers of combiner, reducer, Mongo and MinIO.

vi. Attaching clients to our FEDn system, which in our case are set to be two, but can be configurable as needed. Below we can see the two clients being attached to our only combiner in our FEDn system.

Figure 20. Deploying FEDn into a local machine

The above image shows us the network composition, currently consisting on reducer and combiner, waiting for clients to be joined.

Figure 21. Attaching clients to FEDn system in a local machine
After completing the above step, it is possible to start the training for the desired number of rounds, a timeout range and we can also configure validating or not our models (based on the validate function incorporated into compute package). After starting the training, FEDn UI enables us to see in real time the dashboard with real-time metrics, logs regarding the occurring flow (whether it’s training state, validating state etc.) and most importantly, the validation metrics after the training is done for each round.

The following image display the main FEDn UI dashboard which contains information regarding the mean client processing time for each training round, client training time distribution in histogram and combiner processing time for each round.

![Figure 22. FEDn UI Dashboard](image)

Lastly, in the following image, we can observe the classification metrics for each round used for this example (train/test accuracy, train/test loss), which obviously can be customized accordingly to the nature of the AI model we are using.
The above implementation shows the light and straightforward approach FEDn implies to adapt AI/ML models into their system, and also includes a dedicated tutorial on the structure that should be followed in order to incorporate other custom models.

4.4.2. YOLOv8 Implementation in FEDn

Based on the above example’s structure and the tutorial files in the FEDn GitHub repository, we have built our custom implementation of YOLOv8 model into FEDn Federated Learning framework. Currently speaking, FEDn supports two helper classes that can implement AI-based models, a helper for Keras and one for PyTorch, but obviously, it supports the possibility of building your own helper and utilizing it in FEDn deployment. YOLOv8 model’s architecture is based on a Deep Neural Network, whose foundation is written in PyTorch, so we decided to use PyTorch helper to implement our solution.

The approach of implementing YOLOv8 object detection model is similar to above-mentioned MNIST text recognition model into FEDn, of course with some design changes needed. We started a new project inside the “fedn/examples” folder, called “pytorch-YOLOv8”.

Firstly, considering the data collection procedure we did not need to build any script, since we already had our dataset collection locally. So, we just located the “dataset” folder inside our pytorch-YOLOv8 project.

Secondly, we created a script file (split_data) which takes as a parameter a number N and splits the existing dataset to N pieces, one per each client replica we want to attach. Saying this, if N=2, we
construct a new folder called “data/clients” which contains two directories “1” and “2”, each of them holding 50% of train, valid and test images/labels. If N=3, we would have three directories for each client, each holding 33.3% of the data for each train/valid/test directory, and so on. To conclude, the given script splits the current dataset to 1/N, where N is the number of clients we want our FEDn simulation to have.

![Figure 24. Split dataset tree structure for 3 Clients](image)

**Thirdly**, and the most challenging part, was adapting YOLOv8 to work in coherence with FEDn architecture. Even though YOLOv8 has a PyTorch foundation, its syntax of initializing the model, training it, loading the weights, and validating the resulted model is quite distinct and it does not have any similarities to the PyTorch example. So, we had to improvise. We managed to load a YOLOv8 model in the same way we did when training locally and in Jetson Nano device, with the only difference of saving its output in a “.npz” format, compatible with the PyTorch helper.

Even after solving this matter, another design issue needed to be resolved before moving forward. As mentioned in the previous chapter, our most optimal model of YOLOv8, YOLOv8 M, is a pre-trained model that uses the COCO dataset. When being loaded for the first time, this model contains 80 classes from COCO dataset. When we attempted to load this model and use it to train our custom indoor object detection dataset, we faced mismatch in the Tensor shapes of images. Certainly, our custom dataset required 10 classes and the initial structure of YOLOv8 provided 80 classes.
We were able to surpass this challenge by adding an extra step in our approach: performing a minor training with our custom dataset in order to build an initial model of YOLOv8 with 10 classes and export it as a seed model. After having our initial model, we were able to use it as a starting point for our “train” function in the entrypoint Python script. We were able to train the model, save it in a PyTorch-based version and reload it in the “validate” step, where we can extract training and validation metrics to be displayed in the FEDn UI Model view. By consolidating the “entrypoint” Python script, we were able to execute our custom “build.sh” script and generated the compute package (package.tgz) and our initial model (seed.npz), which are used to configure the deployment of our YOLOv8 model into FEDn.

*Lastly*, we perform the same steps as in the MNIST PyTorch example; we start and run all docker containers (Mongo, MinIO, Reducer, Combiner) and then attach the clients to our combiner, which in our case decided to try with three models locally. The below image displays how the network composition looks like for our YOLOv8 model implementation to FEDn with three clients attached.

![Network composition](image)

*Figure 25. Local YOLOv8 implementation into FEDn with 3 clients*

Each one of the clients has trained their corresponding YOLOv8 model in their part of split dataset, sent the weights to the combiner, the weights has been aggregated and processed in the global model by the reducer and lastly they are being brought back to the clients to
initiate the process of training again (considering the number of rounds attached). Saying this, in the Validation metrics, we decided to customize it according to our evaluation metrics specified in Method chapter: mAP, Precision, Recall, in training and testing processes.

![Figure 26. Evaluating training precision in YOLOv8 implementation to FEDn](image)

Considering the challenges we faced during the implementation part, we can say that our YOLOv8-FEDn system needs further improvements to be fully functional. As we managed to tackle the problem of Tensor shape mismatches between the model stages, we faced other technical issues. Even though YOLOv8 is build based on PyTorch library structure, it has its own wrapper function to train, validate and update weights. This fact caused limitations to our approach, where the FEDn aggregator could not transmit the updated weights correctly to the dedicated clients in some cases. Consequently, training the model using multiple FEDn rounds caused sometimes repetitive results. This is the reason why we state that the state of our system is non-definitive due to these isolated issues.

4.4.3. YOLO-FEDn Deployment

Having established our YOLOv8 implementation into FEDn Federated Learning framework, and having in mind our previous local implementation in local machines and Jetson Nano Kit, the path is ready to continue and deploy our solution into production, a cloud environment where the clients are real edge devices that can be connected remotely from any location in the world. We have used SNIC Science Cloud (SSC), a community cloud platform which offers high-performing computing resources, storage capacity etc., to make
this deployment easier. Using SSC, we can build a distributed eco-
system where edge devices (including Jetson Nano Kit) can take part
in the federated learning process from various geographical regions,
increasing the diversity of the data and resulting in a more reliable
model. This cloud-based implementation allows us to take advantage
of federated learning's full potential by supporting a wider variety of
real-world scenarios and data sources. We initiated the remote clients
be attached to the Combiners via Internet connection, participating in
the training process while maintaining the privacy of their local data.
As stated in the previous chapter, due to isolated issues we had re-
garding the YOLOv8-FEDn implementation, and also the short
amount of time we had at our disposal, we were not able to fully im-
plement our YOLO-FL system in SSC. We were able to set up the
environment and connect the clients to the system, but the technical
problems in our implementation and the limited amount of time did
not allow us to make a full deployment of this system.

5. Results

In this chapter we aim to elaborate on the outcomes of the above im-
plemented methods, sharing our insights on the data gathered and the
results yielded. In the previous implementation chapter, we have al-
ready presented some preliminary results for the object detection
models we have used in different environments and platforms, and
we will use this section to give more details and show them in a tab-
ular representation in order to better visualize the model behavior
and performance in a side-by-side display. In order to construct our
project of Federated Learning for Edge Devices: RTOB, we have
performed several experiments through the path:
- We trained YOLO models in a local machine, where we were able
to choose our optimal option, YOLOv8.
- We trained our YOLOv8 model in a Jetson Nano Kit with
GPU/CPU support options.
- We compared, evaluated between open-source FL frameworks and
chose the optimal one for our use case, FEDn.
- We constructed our YOLOv8 implementation into FEDn system.
- We attempted to deploy our YOLO-FEDn implementation in a
Cloud production environment, but did not manage to finalize it.
Saying this, here is a side-to-side comparison between our object detection models.

<table>
<thead>
<tr>
<th>Model</th>
<th>Platform</th>
<th>Processor</th>
<th>Precision</th>
<th>Recall</th>
<th>mAP (5)</th>
<th>mAP (5-95)</th>
<th>Privacy</th>
<th>Communication</th>
</tr>
</thead>
<tbody>
<tr>
<td>YOLOv7</td>
<td>Local</td>
<td>CPU</td>
<td>0.768</td>
<td>0.373</td>
<td>0.416</td>
<td>0.241</td>
<td>Low</td>
<td>High</td>
</tr>
<tr>
<td>YOLOv8</td>
<td>Local</td>
<td>CPU</td>
<td>0.698</td>
<td>0.620</td>
<td>0.486</td>
<td>0.309</td>
<td>Low</td>
<td>High</td>
</tr>
<tr>
<td>YOLOv8</td>
<td>Jetson</td>
<td>CPU/GPU</td>
<td>0.265</td>
<td>0.202</td>
<td>0.124</td>
<td>0.066</td>
<td>Low</td>
<td>High</td>
</tr>
<tr>
<td>YOLOv8</td>
<td>FEDn</td>
<td>Local</td>
<td>0.342</td>
<td>0.465</td>
<td>0.372</td>
<td>0.198</td>
<td>Moderate</td>
<td>High</td>
</tr>
<tr>
<td>YOLOv8</td>
<td>FEDn</td>
<td>Cloud</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>High</td>
<td>Moderate</td>
</tr>
</tbody>
</table>

Starting with our local models training, we yielded some interesting results, while being able to run many epochs (iteration). The local machine used for training local models is Ubuntu-based machine (Linux), with 16GB memory and a processor of 11th Gen Intel® Core™ i7-1165G7 @ 2.80GHz × 8. Even though the machine has decent parameters for a small laptop, executing these types of models with high number of iterations and batch size, caused a great stretch in the processing time, sometimes up to 24 hours to finish training. The results in our local machine for YOLOv7 and YOLOv8 models reflect some interesting metrics. YOLOv7 has a higher precision but lower recall compared to YOLOv8. This means it's more accurate with its positive predictions but it detects fewer actual positives. While YOLOv8 has much better recall and mAP scores at both IoU=0.5 and the entire range from IoU=0.5 to 0.95, it has slightly inferior precision. This implies that, generally speaking, it performs better at object detection over a range of scenarios and overlaps. If we would want to focus on accuracy only, YOLOv7 would have a slight advantage over the YOLOv8 model and we would choose the first one. But considering the volume of objects we need to detect, Recall and mAP results suggests that YOLOv8 is the better model. We continued training our YOLOv8 model in our edge device, which with many difficulties on setting it up, we were able to train a modest model in its environment. Observing the evaluation metrics, we can clearly see that there is a downgrade in YOLOv8 model training in all of them (Precision, Recall, mAP). This has mostly related to the hardware and software limitations we faced, which are ex-
plained in the Implementation chapter. GPU caused voltage throttling, CPU had a smaller capacity (2GB) and eventually we ran the training with less epochs and less batch size.

Regarding the YOLOv8 implementation into the FEDn system, we can see that there are improvements, comparing to the results coming from Jetson Nano, but still far from YOLOv8 local training. The reason for having these preliminary results in a lower quality is linked to hardware capabilities again. Even though have used the same Ubuntu machine for simulating this YOLO-FEDn integration, we have 2-3 docker containers (clients) running all at once and in the most cases, the training process was killed in the first epoch due to CPU overload. In order to improvise, we had to decrease the number of image batches and epochs so the training can be processed without interruptions. See image below for a visual display of CPU usage while implementing YOLOv8 into FEDn.

![Figure 27. CPU usage while training YOLOv8 in FEDn locally](image_url)

Regarding the YOLO-FEDn integration and deployment into SSC, the above issue is solved. No client needs to run multiple instances in their local machines. A client is being set up to train their models locally while keeping their data private, send their weights to the combiner and re-run the training process when new aggregated weights will be delivered to all the clients. Although this problem was solved, as we previously stated, the YOLO-FEDn integration into SSC cloud was not fully completed. This is why the results were noted as NA (Not Available) in the above results table.
While having in mind that the above results are related mostly to the “A” factor of our APC research problem, we will also note the results on “P & C” factors. As stated in the Method chapter, the evaluation of Privacy and Communication is conducted by analyzing the tools that our chosen FL framework (FEDn) offers, by monitoring how clients coordinate with each other in terms of P & C in different environments. If we have another look at the table of results, we can elaborate the Privacy and Communication performance as follows:

- **Local environment (YOLOv7 & YOLOv8)**, we have noted the privacy to be low, because training an RTOB model in a local machine can be considered training it in a centralized ML environment, where assuming that we would have a lot of stakeholders with their data, we would accumulate the data to train it all in once. We have noted the communication to be high, because there is no hassle on communicating between data inside a centralized system.

- **Jetson Nano environment (YOLOv8)**, we have made the same conclusions in P&C results as above, due to the fact that the model has been trained in the local environment of the toolkit, and not connected to any FL system at this stage.

- **FEDn Local environment (YOLOv8)**, we have noted the privacy to be moderate, due to the fact that we are still being in the same local machine, but we do have a distinct N splits between N number of clients, which makes all of the clients having their unique subset of data. We have noted the communication to be high, because we are still in a local environment and there is no difficulty in communication.

- **FEDn Cloud environment (YOLOv8)**, even though we did not have a complete training result in this platform, we can still talk about P & C factors. We have noted the privacy to be high, considering the fact that each client has their own data in their own local environment and there is no risk of one acquiring the dataset of the other. We have noted the communication to be moderate, considering the potential network issues that clients may have while connecting to Cloud caused by their location, access to the internet, and so on.

Taking into consideration the amount of time we had in our disposal and the hardware computational limitations we faced, we can say
that the above-mentioned results are fairly decent for a trivial Federated Learning application in RTOB at the edge. These results can surely be improved by having higher computational power and a completely implemented YOLO-FEDn system (which will be discussed later in the Future work section).

6. Discussion

We investigated how the selection of our chosen methodologies, including the YOLOv7 and YOLOv8 object detection models, affected the results across various platforms as we delved into the specifics of our findings. Our discussions on the opportunities and difficulties involved in deploying machine learning models on edge devices were guided by the interactions between these methods, the contexts in which they were deployed, and the performance they displayed. The argument was further enriched by our involvement with the Federated Learning framework within the FEDn ecosystem. We looked into the impact of distributed client collaboration on model convergence and performance measures. Our investigation of optimization paths and the closing of performance gaps was shaped by the interaction of hardware constraints, training procedures, and the orchestration of federated learning within this framework. Having in mind multiple experiments across platforms, models and environments, we have built the foundation of being able to exhaust the use cases at our disposal to address the aims and objectives of our study, as well as analyzing the trade-offs between accuracy-privacy-communication. And by specifying that this is a foundation of a FL application in RTOB models at the edge with the available tools we possessed, we can consider a fully-implemented YOLO-FEDn system as a continuation of this project (also mentioned in the Future work section).

6.1. Previous Research

The research background I have been focusing on discusses the recent studies in RTOB in a Federated Learning perspective, FL applications in various AI algorithms, privacy and accuracy concerns in these applications, Cloud and Edge computing applications and lastly FL open-source frameworks and their characteristics.
Considering my research to be a proper literature review in a generic sphere of Federated Learning, Edge Computing and RTOB, the majority of efforts have been allocated to address our research problem in regard to these related work studies. Having observed that FedAvg, the foundation of FL algorithms, is a point of discussions between researchers on whether using it or not for privacy and non-IID data issues, we followed the school that supported its usage with proper adjustments. FEDn offers a communication architecture based on FedAvg, which concludes to a FedAvg-hierarchical structure, which uses a local SGD (Stochastic Gradient Descent) strategy on aggregating weights. This enables support for most of the ANN (Artificial Neural Network) models, SVMs and it is currently in progress for following additional FedML training protocols, which makes FEDn a highly reliable framework and potential for the future. We have based our decision on which FL framework we should focus based on the most relevant FL concepts shared in the background section, while having in mind our Accuracy-Privacy-Connection trade off problem.

Considering the Cloud vs Edge Computing comparison in the background research, we have structured our study in a way that is inclusive for both paradigms. We have been able to train YOLOv8 models locally, we trained the model in an edge device (Jetson Nano) and also we made it possible to combine these edge devices into a cloud platform so they can be trained simultaneously using FEDn framework. The results showed us improvement from edge devices local training to their combination in a cloud platform, so we believe that the combination of these paradigms can be an efficient toolkit to achieve high-performing object detection models.

6.2. Methods, implementation and results

The implementation being done for this project have been thoroughly studied to cover all the necessary steps of a building RTOB model in a FL perspective.

Firstly, the dataset selection was a choice that affected our results drastically. By having different versions of our indoor objects’ detection dataset, we were able to observe the variations in the overall performance of our models. Having a complete dataset and using it for training purposes in different platforms, helped us get better mAP results, but drastically affected the processing computation usage and


the execution time; both were increased significantly. By splitting
the dataset in N number of clients when testing the YOLO-FEDn im-
plementation locally, we were able to enhance the processor usage
and yield faster results, but the models’ performance was noticeably
dowgraded. Various preprocessing and augmentation techniques
(grayscale, resize, flip, rotate) have been applied while training our
RTOB model in different platforms by using Roboflow tool, which
resulted in better processor usage and smoother training process.
Secondly, by selecting YOLOv8 model over YOLOv7, we managed
to get better overall performance results in almost all the platforms
where we trained these models. Selecting YOLOv8 had its own costs
in terms of dedicated time we invested to solve the implementation
bugs in Jetson Nano Kit and also in FEDn. Being the latest technol-
ogy in YOLO family, YOLOv8 contained highly upgraded Python
libraries which were not compatible with FEDn and Jetson Nano by
default. Necessary adjustments have been done to adapt YOLOv8 in
these environments.
Thirdly, the selection of the edge device for this project has been
quite a challenging one, since it brought many operating system (OS)
bugs during its installation. Jetson Nano, being a toolkit, which had
reached its End-Of-Life (EOL) and had no longer support for latest
libraries, and in the other hand YOLOv8 being the latest word of
technology regarding YOLO family, were understandably not easy to
adapt with each other. Consequently, the training results achieved
while training YOLOv8 model locally were not the most optimal that
we expected, considering we could not fully utilize its computing
power.
Lastly, the decision of picking FEDn as our FL framework impacted
our results in a good trajectory, performance-wise speaking. Even
though this implementation raised a lot of CPU overloading when
implemented locally, it can gradually be improved when deployed in
Cloud.

Generally speaking, the selected methods and implementation means
have helped us gradually improve the performance of the model and
continuously observe the trade-off we face between accuracy, com-
putation power, privacy and communication. The trade-off between
APC factors in various platforms, as described in the Results chapter,
proved us that is difficult to maintain a balance between them, where
having a higher result in one factor means deprecating the other.
- **Accuracy** turned out to be better in local environment for YOLOv8 model, with FEDn local implementation having a slight improvement from Jetson Nano, but yet not close to local YOLOv8 model. We expected to have better results while deploying FEDn into Cloud, but we were not able to prove it, considering the limitations we had in this matter.
- **Privacy** turned out to improve while moving from local environments (where we had no clients at all with a common dataset), to FEDn local (where we have N clients with N subsets of the same dataset), to FEDn in Cloud (where each client had its own dataset in their local environment).
- **Communication** turned out not to be a problem when training YOLO models locally, since we did not need any interaction between clients in these environments, but it started to deprecate when moved to FEDn Cloud considering clients’ connectivity issues.

We believe that our preliminary results and methodic are a decent foundation for other researchers to work on similar domain. A broad application of insights can be accomplished by analyzing our obtained results. While the specifics of the datasets, models, and platforms may differ, the trade-offs between accuracy, computation power, privacy, and communication are relevant considerations in various scenarios. Our results highlight the significance of dataset quality, model choice, and implementation details in real-world tasks like real-time object recognition on edge devices.

### 6.3. Ethical and societal aspects

Our project has significant implications for societal and ethical concerns. Privacy issues emerge as we examine the use of data for training, particularly when considering the presence of private data in object detection datasets. While we have carefully chosen a dataset that does not hold sensitive information for any particular person or organization, the potential for unintended inferences remains, which emphasizes the value of effective data privacy safeguards. Some of the potential ethical and societal concerns that can emerge from our project are as following:
- **Bias and Fairness**: The data collected from different clients (edge devices) in our YOLO-FEDn implementation may not be representative of the entire group of objects in indoor environments, leading to
biased models that perform well on some subsets but poorly on others. In regard to what context, we are utilizing our indoor detection model, different classes and/or images can be used. Ensuring fairness in object detection models is vital to prevent reinforcing existing societal biases.

- **Model Robustness**: High accuracy is required for deploying object detection models on edge devices, especially for safety-critical applications. When models can't reliably recognize objects, there are ethical issues that could result in accidents or injury, especially considering our suggestions for healthcare applications.

- **Malicious Attacks**: As mentioned in the background chapter, even with security protocols activated and secure communication between parties (that FEDn already provides), the probability of malicious attacks exists. A high importance should be shifted to computing packages, model updates communication between parties, especially combiners and clients, where the malwares can be injected into these files and disrupt the framework and consequently risk the local data of each client.

- **Social Impact**: It is possible that people’s perception and beliefs towards RT-OB models change, if they are widely used in edge devices. Potential changes to privacy standards, altered perceptions of surveillance, and the effects of more automation on human decision-making can all be labeled as ethical issues to society.

7. Conclusion

We have pursued a thorough study on Federated Learning on Edge Devices, into a Real-Time Object Detection perspective. We assessed, analyzed and learned through the process, by picking the most optimal approaches and tackling challenges that occurred. We were able to start from testing locally a few versions of YOLOv7 and YOLOv8 models, continuing to set up and configure our edge device which was taken into account (Jetson Nano 2GB Developer Kit), evaluated and choose between FL open-source framework by studying their infrastructure and how their features fitted to our aims and objectives and lastly creating a non-fully completed YOLO-FEDn implementation and training our edge clients in a production environment. Saying the above, we can categorize our conclusions into three main categories:
Aims & Objectives:
- After carefully investigating the related work and the developments that have been done into the usage of FL for real-time object detection at the edge, we were able to grasp architecture, privacy, accuracy concepts, choose from the most used FL algorithms, use the most optimal FL framework (FEDn) to implement an RTOB-FL solution in Edge Computing.
- Trade-offs between accuracy, privacy and communication has been carefully studied from previous research and also been put to practice by training YOLOv8 model into multiple environments and measuring these variations in each one.
- Even though our YOLO-FEDn implementation in Cloud was not fully integrated, we can say that the preliminary results showed a plausible balance between APC factors. Accuracy is expected to have improvements from the local environments; Privacy is established and each client has their own data in their local machine; Communication so far is moderately good, having minor latencies, due to the connectivity and different nature of the edge devices connected.

Research Questions:
- How can FL be utilized for training shared model for real-time object detection at the edge?

Federated Learning is the adequate technique of training a shared model between edge devices for real-time object detection purposes and not only. This can be achieved by evaluating the current developments that has emerged in FL sphere, choosing the proper FL framework to the needs of the RTOB problem and adapt the model to that particular framework. The way that we did tackle our approach to utilize FL for RTOB at the edge was using FEDn open-source framework and adapt YOLOv8 model into this system, by using edge devices (clients) such as Jetson Nano Kit to train the model.
- What are the trade-offs between accuracy, privacy, and communication overhead in this sphere?

By having various platform to develop our solution (local machines, Jetson Nano Kit, VMs, Cloud service) and different models (YOLOv7 and YOLOv8 M,N,X), we were able to see in action how these trade-offs took place.
As previously described in the Results chapter, when implementing the solution in a local machine, even though the accuracy was decent and the communication was easy, privacy factor is low, which is an expected result due to the fact that all the clients were simulated to be run into a single machine and consequently each one could gain access to the other one’s dataset.

When implementing the solution to Jetson Nano Kit only, we had a decrease to all the APC factors due to technical and hardware challenges with the device.

When implementing the solution into a production YOLO-FEDn environment, even though not in its final state, we were able to explore a new insight that the privacy is improved (each client having their dataset locally), but we had a drawback in the communication overhead caused by the different nature of the edge devices and the internet connectivity.

- **How can we improve the performance of FL for real-time object detection?**

The overall performance of an FL system for real-time object detection can be improved by finding a balance between the above-mentioned factors: accuracy, privacy and communication. We can achieve this by having a high computation server that will host the YOLO-FEDn system in the cloud, having homogeneity between edge devices so we avoid communication issues due to changes in nature of the clients and also applying extra security layers like encryption when communicating between combiners and clients, to avoid the risk of being exposed to malicious attacks.

**Results:**
- We experienced a growth in model performance while moving from individual training in local machines to a collaborative environment.
- Our most optimal RTOB model resulted to be YOLOv8 Model M.
- Our most optimal FL framework resulted to be FEDn.
- FEDn resulted to be an optimal approach for our case, by also considering privacy matters, having applied SSL/TLS protocols of security in the requests between combiners and clients.
- We faced many issues in Jetson Nano Kit and its implementation with YOLOv8 model. A potential improvement would have been
choosing a more updated toolkit to support modern libraries like YOLOv8.
- The implementation of YOLO-FEDn was not fully completed in Cloud due to an isolated technical issue with YOLOv8 wrapper and the limited amount of time we had at our disposal.
- FEDn implementation had its own challenges while setting it up locally and also in the cloud, but once we fully comprehended the system’s structure, the logic was easy to be followed.
- Our project’s goal can be considered an achieved one, since we were able to build a real-time object detection implementation using FL on the edge and also addressed our research questions by following a methodology constructed around APC concepts.
- We were able to gain significant knowledge and experience through the whole process, by putting in life the concepts we have learned in this Master program, but also the new ones we learned by doing and facing various challenges.

7.1. Future work

As previously mentioned, our project did not conclude the way we wished to, where we had issues on fully deploying the YOLOv8 implementation into FEDn Cloud platform. In a project-oriented perspective, this thesis work has place for continuation in regard to the technical issues we faced with YOLOv8 wrapper and establishing weights aggregation when communicating to FEDn framework. By accomplishing this, we can continue training YOLO-FEDn model locally and in cloud and have a full picture about its efficacy.

On the other hand, researchers working in related fields can learn from our experiences by adapting their decisions to their own situations. The place where the opportunities arise for other researchers, lies in refining edge device deployments, optimizing model convergence through federated learning, and striking the right balance between performance and resource constraints.

In a broader picture, although we believe that our work has been quite relevant in Federated Learning, Edge Computing and Real-Time Object Detection domains, we acknowledge that it needs improvements and can be a good genesis for potential other research continuation.
- **Accessibility for the Visually Impaired**: The chosen dataset is part of a project for obstacle detection for blind people, and saying this, we believe that this project has a great potential to extend the RTOB models on edge devices to enhance the accessibility for the visually impaired individuals. Once text-to-speech functionality is incorporated into the deployed models, blind people can navigate and engage with their environment more successfully since they will have valuable audio information about their surroundings. Another topic of research in this direction would be considering privacy implications and user data security when developing such features, and investigate how federated learning may be used to increase the precision and tailor the text-to-speech functionality across various edge devices and user circumstances.

- **Dynamic Resource Allocation**: Even though we proposed that one of the factors which can facilitate achieving high-performing FL systems for RTOB, would be having homogeneous edge devices, we should admit that such cases are rarely to happen in the real world, where we can face edge devices of different hardware/software configurations. In order to dynamically distribute computing resources among edge devices based on their processing capabilities and current workloads, potential research can be initiated to investigate adaptive resource allocation algorithms. This could optimize communication overhead and model convergence rates while maintaining privacy.

- **Resilient Training with Cached Models**: Not every edge device has a stable internet connectivity or speed and this may affect the performance of our RTOB model in an FL system. A potential future research can address the challenges of intermittent or inconsistent internet connectivity in FL environments by experimenting with methods to use cached models and training data on edge devices. Methods to investigate strategies of detecting network interruptions and storing intermediate model updates and training progress, can be studied. Another relevant factor to consider is developing algorithms that allow for edge device training and collaboration with FL frameworks using cached data, and in the same time being able to synchronize cached updates with the central server once connectivity is restored.
References


