Federated Learning for Heterogeneous Object Detection

submitted in partial fulfillment of the requirements

for the degree of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINNEERING

by

AAKARSH BANSAL **CS22B001**

JAIMIN VIRAMGAMA **CS22B051**

Supervisor(s)

Dr. Vishnu Chalavadi

भारतीय प्रौद्योगिकी संस्थान तिरुपति

TIRUPATI

DEPARTMENT OF COMPUTER SCIENCE AND ENGINNEERING

INDIAN INSTITUTE OF TECHNOLOGY TIRUPATI

OCTOBER 2025

DECLARATION

We declare that this written submission represents our ideas in our own words and where others' ideas or words have been included, we have adequately cited and referenced the original sources. We also declare that we have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in our submission to the best of my knowledge. We understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

Place: Tirupati	**Signature**	**Signature**
Date: 29-11-2025	Jaimin Viramgama	Aakarsh Bansal
	CS22B051	CS22B001

BONA FIDE CERTIFICATE

This is to certify that the report titled **Federated Learning for Heterogeneous Object Detection**, submitted by **Author(s)**, to the Indian Institute of Technology, Tirupati, for the award of the degree of **Bachelor of Technology**, is a bona fide record of the project work done by him [or her] under our or [my] supervision. The contents of this report, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Place: Tirupati
Date: 29-11-2025

Dr. Vishnu Chalavadi
Guide
Assistant Professor
Department of Computer Science Engineering
IIT Tirupati - 517501

ACKNOWLEDGMENTS

The authors thank the Department of Computer Science and Engineering, IIT Tirupati, for providing hardware and laboratory resources. Special thanks to our project supervisor, **Dr. Vishnu Chalavadi**, for continuous guidance and valuable insights

ABSTRACT

KEYWORDS: Federated Learning; Object Detection; Format.

Federated Learning (FL) provides a privacy-preserving approach to distributed model training where multiple clients collaborate without sharing their raw data. This work presents the progress of the entire B.Tech project aimed at implementing a federated learning system for heterogeneous object detection using YOLOv3. Two Raspberry Pi devices act as FL clients and a laptop serves as the central aggregation server. Each client trains its YOLOv3 model locally on a subset of a custom dataset and contributes updates for global aggregation using the FedAvg algorithm. The system successfully establishes client–server communication, performs independent training, and conducts aggregation. Current challenges include parameter mismatches between client models during aggregation, resulting in incompatibility during model loading and prediction.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS

The research scholar/ student must take utmost care in the use of technical abbreviations. For example, gms stands for gram meter second not grams. Grams should be abbreviated as g. Abbreviations should be listed in alphabetical orders as shown below.

API	Application Programming Interface
AHA	Adaptive Hierarchical Aggregation
CNN	Convolutional Neural Network
CPU	Central Processing Unit
FL	Federated Learning
FLWR	Flower Federated Learning Framework
gRPC	Google Remote Procedure Call
IID	Independent and Identically Distributed
IoT	Internet of Things
ML	Machine Learning
MOON	Model-Contrastive Federated Learning
NLP	Natural Language Processing
NN	Neural Network
Non-IID	Non–Independent and Non–Identically Distributed
OS	Operating System
RAM	Random Access Memory
REST	Representational State Transfer
RPI	Raspberry Pi
SGD	Stochastic Gradient Descent
ZSL	Zero-Shot Learning
YOLO	You Only Look Once

NOTATION

The research scholar/student must explain the meaning of special symbols and notations used in the thesis. Define English symbols, Greek symbols and then miscellaneous symbols Some examples are listed here.

K	Total number of clients in federated learning
n_k	Number of training samples on client k
n	Total number of samples across all clients ($n = \sum n_k$)
w_k	Model weight vector of client k after local training
w^t	Global model weights at communication round t
w^{t+1}	Updated global model weights after FedAvg aggregation
L	Loss function of the YOLOv3 model
x	Input image to the object detection model
y	Ground-truth labels (bounding boxes and class IDs)

Greek Symbols

η	Learning rate during local training
θ	General notation for trainable model parameters
σ	Confidence score for predicted bounding boxes

Miscellaneous Symbols

B	Mini-batch of training samples
E	Number of local epochs per communication round
λ	Regularization or weighting coefficient (if applicable)
IoU	Intersection over Union (evaluation metric)

CHAPTER 1

INTRODUCTION

This chapter provides an overview of the project, its motivation, and objectives. It introduces the concept of federated learning and outlines the scope of work undertaken in this study.

1.1 Problem Description

Deep learning–based object detection models such as YOLO have demonstrated outstanding performance in visual perception tasks. However, training these models generally requires centralizing large datasets, leading to concerns over privacy, latency, and ownership. Federated Learning (FL) mitigates these challenges by enabling distributed clients to collaboratively train a global model without sharing raw data.

This project focuses on developing a federated object detection system for heterogeneous devices using YOLOv3. The setup involves two Raspberry Pi clients and a laptop acting as the central aggregation server. Each client trains a local YOLOv3 model using a portion of a custom dataset, and the global model is periodically updated via the *FedAvg* aggregation algorithm.

1.2 Objectives and Scope

The primary objectives of this project are:

- To design and implement a working federated learning framework using heterogeneous edge devices.
- To apply the framework for object detection using YOLOv3.
- To evaluate the feasibility and performance of FL in constrained environments such as Raspberry Pis.
- To identify and address challenges arising from non-identical model structures or computational limitations.

1.3 Organization of the Report

The rest of this thesis is organized as follows:

- Chapter 2 presents a brief literature review and discusses prior works relevant to federated learning and edge-based object detection.
- Chapter 3 describes the methodology, experimental setup, and mathematical formulations used.
- Chapter 4 discusses the results obtained and observations made during experimentation.
- Chapter 5 concludes the report and outlines future directions.

CHAPTER 2

Literature Review

This chapter presents a brief review of existing research and technologies relevant to the development of a federated learning–based heterogeneous object detection framework. It also outlines the architectural context within which the present work is carried out.

2.1 Overview of Federated Learning

Federated Learning (FL) is a distributed machine learning paradigm in which multiple clients collaboratively train a global model under the coordination of a central server while keeping the training data localized. This framework preserves privacy by transmitting only model parameters or gradients instead of raw data. Popular aggregation techniques include *Federated Averaging (FedAvg)*, *FedProx*, and *FedNova*, among others.

FL has found applications in domains such as mobile keyboards, healthcare, and IoT networks, where privacy and bandwidth are major constraints. However, deploying FL in heterogeneous systems—where participating devices differ in computation power, storage, and network capabilities—remains a key challenge.

2.2 Communication-Efficient Federated Learning Approaches

A key limitation in Federated Learning (FL) is the communication overhead caused by frequent model exchanges between clients and the server, which *“is only exacerbated by the immense size of modern transformer-based models”* . Traditional methods such as the FEDOPT family use a fixed number of local updates before aggregation, but this creates a trade-off where *“lower values may improve convergence but skyrocket communication overhead, while higher values reduce communication cost but risk unpredictable training behavior or non-convergence”* .

To improve efficiency, Federated Dynamic Averaging (FDA) adapts communication frequency by monitoring model variance during training and aggregating only when instability appears, thus reducing communication significantly . However, FDA requires strict synchronization and careful threshold tuning. The recent FDA-OPT approach addresses these issues and *“delivers superior performance out of the box”* as a *“drop-in replacement for FEDOPT”* , achieving *“at least 2× communication-efficiency and 5×–10× lower training loss”* under the same settings . This makes FDA-OPT well-suited for resource-constrained FL environments like the heterogeneous edge devices explored in this work.

2.3 Hierarchical Aggregation in Federated Object Detection

Federated object detection often faces challenges due to heterogeneous device capabilities and non-IID data distributions, which can lead to unstable convergence and poor global performance. To mitigate these issues, hierarchical aggregation has been explored as a scalable communication architecture. This strategy enables clients with similar characteristics—such as geographic proximity or sensing similarity—to first aggregate among themselves before contributing to the global model. The paper emphasizes that this approach is particularly beneficial because real-world object detection deployments commonly involve “weak or noisy local detection labels”, which makes direct aggregation less reliable in unconstrained environments. By structuring aggregation in local tiers, the method improves robustness when label quality varies between clients.

The proposed Adaptive Hierarchical Aggregation (AHA) framework further enhances this design by dynamically selecting intermediate aggregation heads across multiple layers of the hierarchy, based on label quality, model performance, and system heterogeneity. The authors report that this approach improves overall accuracy and stability, stating that “AHA robustly improves over the baseline in all experiment configurations, highlighting its scalability and robustness to federated object detection with weak or noisy local labels.” These findings show that hierarchical and adaptive aggregation strategies are well-suited for federated learning scenarios involving diverse edge devices—aligning directly with the goals of this work on heterogeneous object detection deployment.

2.4 Model-Contrastive Federated Learning (MOON)

Handling data heterogeneity remains one of the core challenges in federated learning, as skewed client datasets lead to local updates that diverge from the global optimization goal. Existing approaches such as FedProx and SCAFFOLD aim to correct this drift during local training, yet they often fail to deliver significant improvements on deep learning–based image tasks under strong non-IID settings. The Model-Contrastive Federated Learning framework (MOON) introduces a novel perspective to this problem by enforcing consistency between global and local model representations. The authors argue that "the global model trained on a whole dataset is able to learn a better representation than the local model trained on a skewed subset", highlighting that local model drift degrades representation quality in non-IID environments.

To mitigate this issue, MOON performs contrastive learning at the model level by maximizing similarity between the current local model's representations and those of the global model, while increasing the distance from the previous local model's representations. This simple modification to the FedAvg local objective significantly improves robustness to heterogeneous data. Extensive experiments demonstrate that "MOON significantly outperforms the other state-of-the-art federated learning algorithms on various image classification tasks", improving accuracy by at least 2% on CIFAR-10, CIFAR-100, and Tiny-ImageNet datasets while maintaining efficient convergence. The study establishes model-based contrastive learning as an effective strategy for federated learning with deep models under real-world non-IID conditions.

2.5 Related Work on Object Detection and YOLO Models

Traditional object detection frameworks such as R-CNN, Fast R-CNN, and Faster R-CNN rely on two-stage detection mechanisms, whereas the YOLO (You Only Look Once) family of models employs single-stage detection that enables real-time inference with reasonable accuracy. YOLOv3 in particular strikes a balance between speed and accuracy by employing Darknet-53 as the backbone and using multi-scale predictions. It remains one of the most widely adopted models for edge-device deployment due to its computational efficiency.

Prior works on federated object detection have primarily focused on centralized GPU clusters or homogeneous edge devices. The present work differs by emphasizing heterogeneous clients—specifically Raspberry Pis—with varying hardware capabilities.

2.6 System Architecture

The implemented system follows a client–server architecture where:

- **Clients:** Two Raspberry Pi 4B devices perform local training of the YOLOv3 model using respective dataset partitions.
- **Server:** A laptop serves as the central aggregation node, coordinating communication and model updates.
- **Network:** All devices communicate over a common Wi-Fi network to exchange parameters.

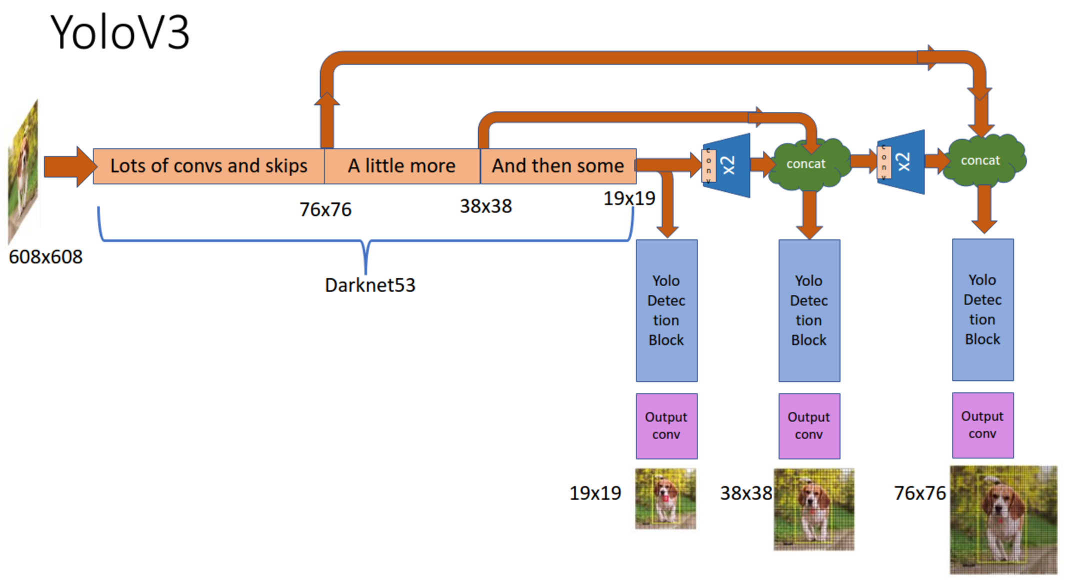


Figure 2.1: Federated learning system architecture showing clients and server communication.

The overall architecture ensures that data remains local to each client while model updates are transmitted to the server. This design aligns with the privacy-preserving principle of FL and reduces data transfer overhead.

2.7 Summary

This chapter reviewed the theoretical foundations of federated learning and object detection, emphasizing related research that supports the current work. It also provided a high-level overview of the system architecture designed for this project. Detailed implementation methodology, including experimental setup and mathematical formulation, is discussed in Chapter 3.

CHAPTER 3

Methodology and Implementation

This chapter describes the detailed methodology followed in the development and experimentation of the federated object detection system. It outlines the hardware and software setup, mathematical formulation, and workflow adopted for implementing the federated learning framework using heterogeneous clients.

3.1 About this Chapter

The purpose of this chapter is to present the design principles, implementation choices, and technical formulation underlying the project. This includes hardware and software configurations, communication mechanisms between clients and server, the FedAvg aggregation algorithm, and a discussion of experimental procedures and parameters.

3.2 Hardware and Software Setup

3.2.1 Hardware Configuration

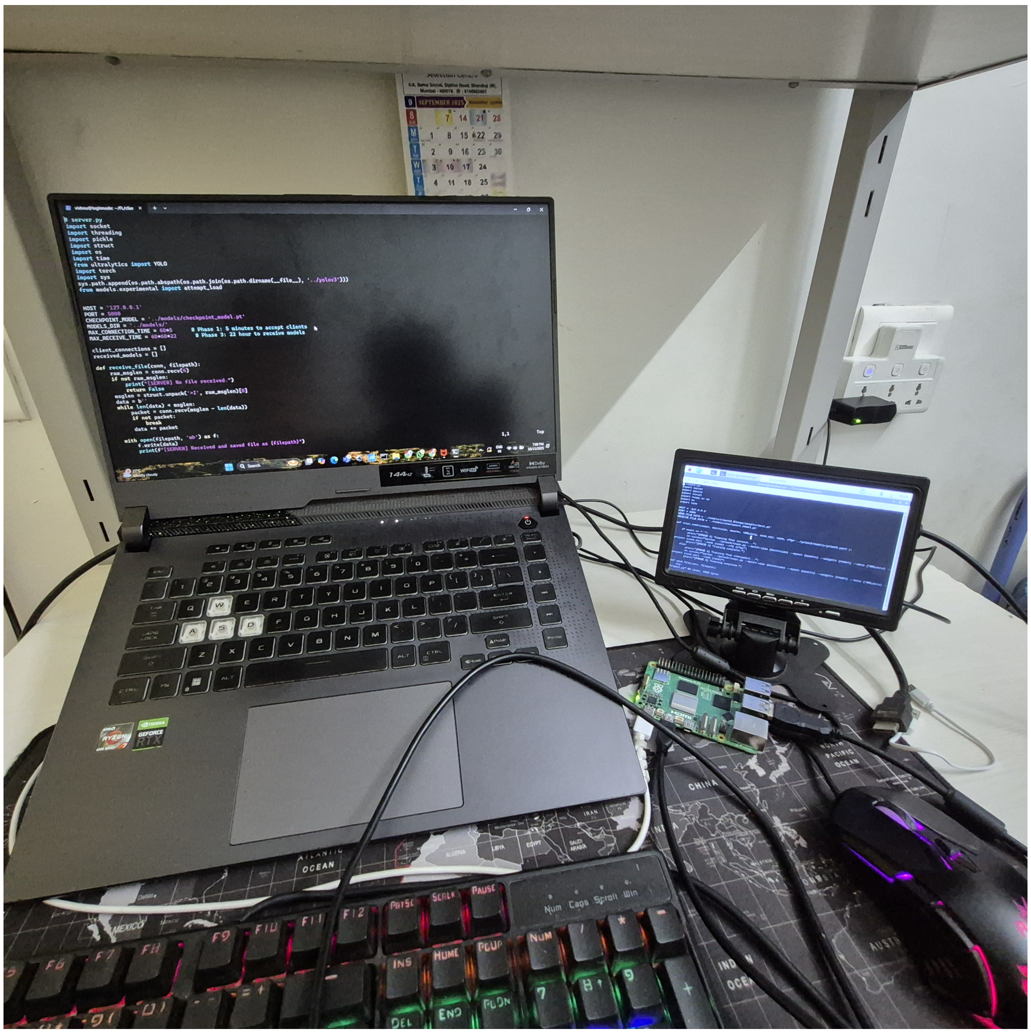
The following hardware setup was used for the implementation:

- **Clients:** Two Raspberry Pi 5 (64-bit quad-core, 8GB RAM)
- **Server:** Laptop with AMD Ryzen R7800H Processor and 16GB RAM
- **Network:** Local IIT Tirupati Wi-Fi network

Each Raspberry Pi device was configured with Raspbian OS and equipped with the necessary dependencies such as Python 3.9, PyTorch, NumPy, OpenCV, and Flask. These clients act as federated nodes responsible for local model training.

3.2.2 Software Stack

The server node manages client registration, weight aggregation, and global model distribution. The overall communication is handled using REST APIs built on Flask. The local models are trained using YOLOv3, with weight synchronization carried out periodically after every few local epochs.



```
server.py
import socket
import threading
import pickle
import struct
import os
import time
from ultralytics import YOLO
import torch
import sys
sys.path.append(os.path.abspath(os.path.join(os.path.dirname(__file__), '../yolov3')))
from models.experimental import attempt_load


HOST = '127.0.0.1'
PORT = 5000
CHECKPOINT_MODEL = '../models/checkpoint_model.pt'
MODELS_DIR = '../models/'
MAX_CONNECTION_TIME = 60*5       # Phase 1: 5 minutes to accept clients
MAX_RECEIVE_TIME = 60*60*22      # Phase 3: 22 hour to receive models

client_connections = []
received_models = []

def receive_file(conn, filepath):
    raw_msglen = conn.recv(4)
    if not raw_msglen:
        print("[SERVER] No file received.")
        return False
    msglen = struct.unpack('>I', raw_msglen)[0]
    data = b''
    while len(data) < msglen:
        packet = conn.recv(msglen - len(data))
        if not packet:
            break
        data += packet

    with open(filepath, 'wb') as f:
        f.write(data)
        print(f"[SERVER] Received and saved file as {filepath}")
```

Figure 3.1: Hardware setup showing Raspberry Pi clients and central laptop server.

3.3 Experimental Setup and Parameters

Table 3.1 summarizes the key hyperparameters and experimental configuration adopted in this study.

Table 3.1: Experimental Configuration and Hyperparameters

Parameter	**Value / Description**
Model Architecture	YOLOv3
Framework	PyTorch
Training Rounds	1 (Mid-term phase)
Local Epochs per Round	5
Batch Size	16
Learning Rate	0.001
Optimizer	Adam
Loss Function	YOLO Object Detection Loss

Each client trained its model locally and sent the updated weights to the central server for aggregation. The aggregated weights were then redistributed back to the clients for subsequent training iterations.

3.4 Workflow

The workflow followed during the implementation is summarized in Figure 3.2. It illustrates the sequence of operations from initialization to aggregation.

1. The server initializes and distributes the global YOLOv3 model to all clients.
2. Each Raspberry Pi client trains the local model on its assigned dataset for a fixed number of epochs.
3. Clients send their trained weights to the server via REST API.
4. The server performs weight aggregation using FedAvg.
5. The updated global model is then redistributed to the clients.

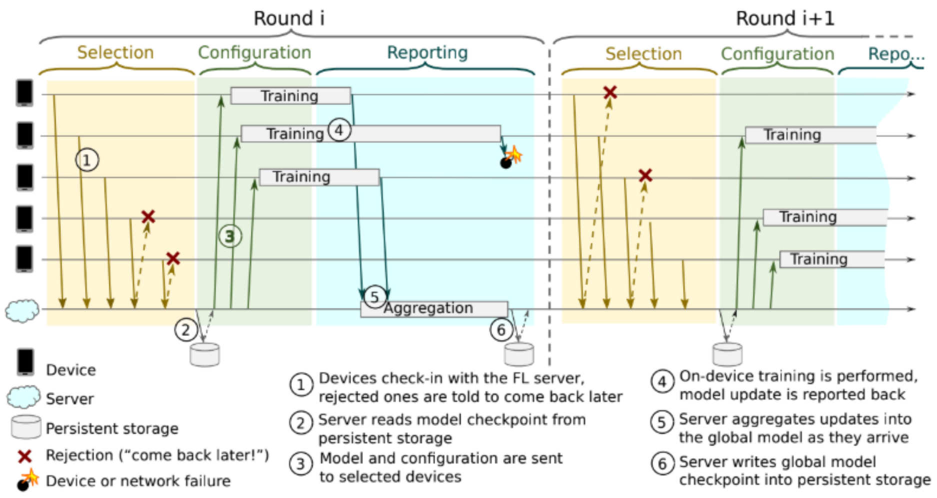

Round i
Round i+1
Selection
Configuration
Reporting
Selection
Configuration
Repo...
Training
Training
Training
Aggregation
Training
Training
Training
Device
Server
Persistent storage
Rejection ("come back later!")
Device or network failure
1 Devices check-in with the FL server, rejected ones are told to come back later
2 Server reads model checkpoint from persistent storage
3 Model and configuration are sent to selected devices
4 On-device training is performed, model update is reported back
5 Server aggregates updates into the global model as they arrive
6 Server writes global model checkpoint into persistent storage

Figure 3.2: Workflow of the federated learning training and aggregation process.

3.5 Additional Implementation Attempts & Architectural Challenges

During the course of implementation, the initial custom-built federated learning framework revealed a critical issue: the aggregated global model exhibited architectural inconsistencies, which prevented the final YOLOv3 model from performing object detection on test images. To ensure that this issue was not a result of our custom system design, we systematically explored alternative approaches and implementation pathways. This section outlines these additional attempts and evaluates their feasibility and outcomes.

3.5.1 Custom Federated Learning Implementation (Current System)

The primary federated learning framework used throughout this project was fully developed from scratch without relying on any external federated learning libraries. This custom architecture handles:

- Client–server communication over a shared network,
- Serialization and transfer of weight tensors,
- Local YOLOv3 training on each client,
- Federated aggregation (FedAvg) on the central server.

Despite successfully enabling communication and training, the aggregated model consistently exhibited parameter misalignment. Even though all clients initialized from an identical YOLOv3 checkpoint, slight variations in training caused mismatched state-dictionary structures, resulting in a global model that failed to produce detections.

This observation prompted further investigation using alternative implementations to validate whether the issue was inherent to our system or a broader architectural challenge with YOLOv3 in a federated setting.

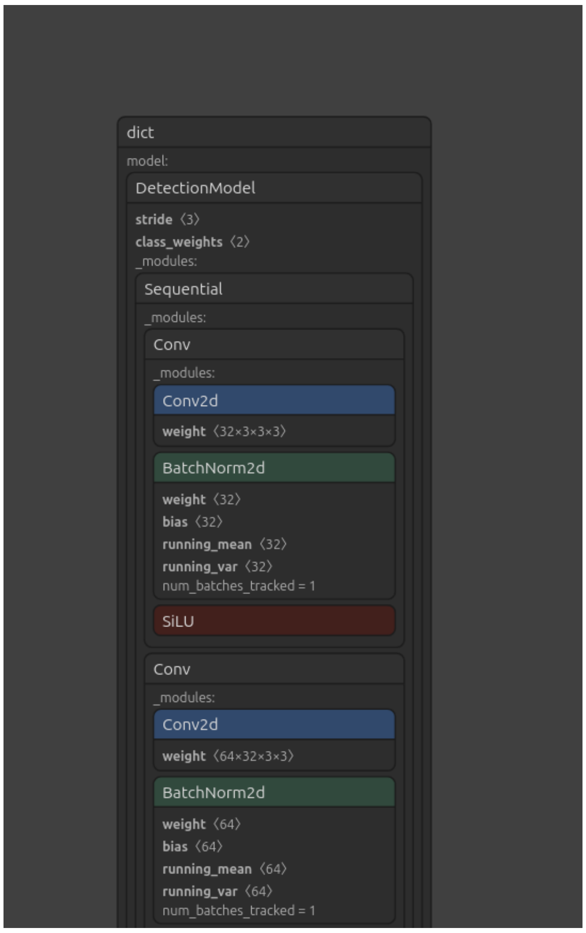


Figure 3.3: The layer-wise architecture for Individual YOLOv3 retrained client models

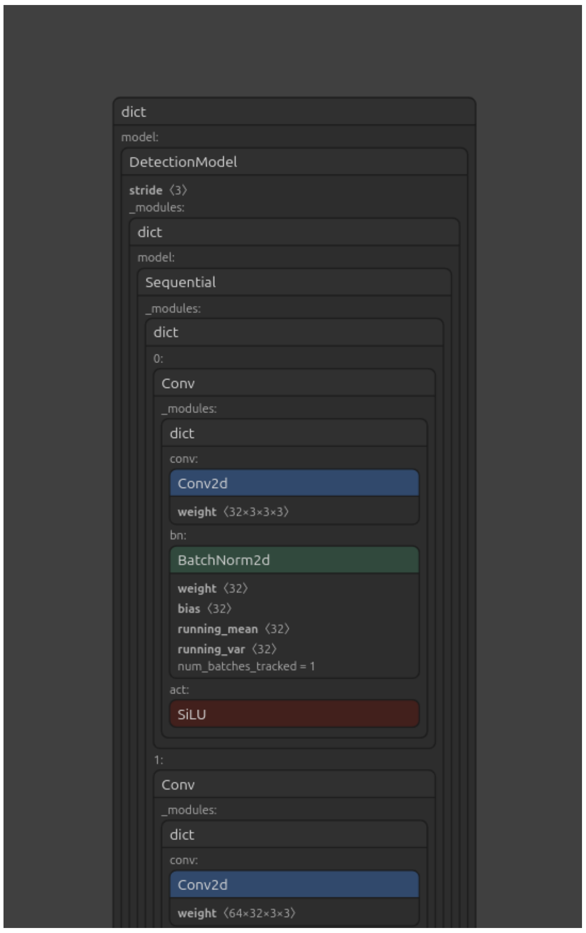


Figure 3.4: The layer-wise architecture for aggregated (manual or using Flower) models

Mathematical Formulation

The global model parameter aggregation is performed using the Federated Averaging (FedAvg) algorithm. Let w_k denote the model weights of client k, and n_k denote the number of training samples in client k's local dataset. The global model update at round $t+1$ is computed as:

$$w_{t+1} = \sum_{k=1}^{K} \frac{n_k}{n} w_k$$

where $n = \sum_{k=1}^{K} n_k$. This formulation ensures that clients with larger datasets contribute proportionally more to the aggregated model.

The update process continues iteratively until convergence or until the predefined number of communication rounds is reached.

3.5.2 Implementation Using the Flower Federated Learning Framework

To cross-verify our findings, we attempted a full reimplementation using **Flower (FLWR)**, a widely used open-source federated learning library designed to abstract complexities related to communication, orchestration, and aggregation.

Rationale

Flower automates:

- Client registration and orchestration,
- Weight exchange protocols,
- Aggregation logic (including FedAvg),
- Error handling in communication.
- Utilizes gRPC protocol(based on json) for faster and reliable communication

This made it an ideal benchmark to test whether the aggregation mismatch in our custom system was due to design flaws in communication/serialization or a deeper architectural issue.

Outcome

Despite Flower's robust infrastructure, **the aggregated YOLOv3 model again resulted in the same architectural mismatch**, ultimately producing a non-functional global model that could not detect objects.

Inference

This result confirmed that:

- The issue is **not** caused by our custom communication or aggregation logic,
- The problem stems from inherent architectural sensitivity in YOLOv3, especially when trained independently on heterogeneous devices with varying computation patterns.

Thus, Flower-based FL was deemed unsuitable for resolving the architecture mismatch for YOLOv3 in this project.

3.5.3 Exploration of Zero-Shot Learning for Federated Object Detection

During the literature study, we evaluated the feasibility of incorporating **Zero-Shot Learning (ZSL)** as an alternative solution. ZSL is designed for tasks where models must recognize classes they have not been directly trained on.

Rationale

The idea was investigated to determine whether ZSL could alleviate dependence on tightly synchronized supervised learning models in federated environments.

Findings

Zero-shot learning is fundamentally built upon:

- Unsupervised or weakly supervised semantic representation learning,
- Transfer of knowledge through attribute or embedding space mappings.

In contrast, YOLOv3 and object detection pipelines rely on:

- Strictly supervised training,
- Bounding-box regression and confidence-layer optimization,
- Pixel-level annotation consistency.

Conclusion

Given that ZSL and YOLOv3 are based on entirely incompatible training paradigms, ZSL is **not applicable** in this problem domain. Therefore, it was excluded from the implementation pipeline.

3.5.4 TensorFlow-Based Reimplementation Attempt

A third direction explored was replacing the entire PyTorch-based YOLOv3 pipeline with an equivalent **TensorFlow** implementation.

Motivation

The hypothesis was that the mismatch in aggregated weights could be related to internal PyTorch-specific model state representations or tensor alignment mechanisms.

Challenges

- The existing system (dataset loaders, training scripts, and custom FL code) was deeply coupled with PyTorch,
- YOLOv3 TensorFlow variants differ in layer definitions, weight formats, and pretrained checkpoint compatibility,
- Porting the entire federated pipeline to TensorFlow would require a near-complete re-engineering of the project.

Conclusion

Due to the extensive overhead and architectural differences, migrating to TensorFlow was deemed **not feasible within the project scope**.

3.6 Summary

This chapter detailed the methodological framework adopted in the implementation of the federated learning system, including the hardware configuration, software setup, mathematical formulation, and workflow. Further the investigation of alternative implementations validated that the architectural mismatch in YOLOv3 is a fundamental challenge when performing federated aggregation across heterogeneous training environments. In particular:

- Flower-based FL reproduced the same mismatch, confirming the issue is model-specific rather than system-specific,
- Zero-shot learning is incompatible with YOLOv3's supervised training requirements,
- TensorFlow reimplementation was impractical and beyond the feasible scope of the project.

These explorations strengthened the insight that further work must focus on parameter alignment, layer-freezing strategies, or architecture-invariant model reformulations to achieve a functional federated object detection system. The next chapter presents the results obtained and provides a discussion on the observed system behavior.

CHAPTER 4

Results and Discussions

This chapter presents the experimental outcomes obtained from the federated learning setup and discusses the corresponding observations. The performance and challenges encountered during implementation are also elaborated.

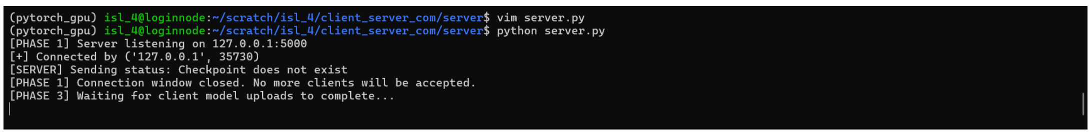
4.1 Overview

The progress of the entire B.Tech project demonstrates the feasibility of implementing a practical federated learning framework for heterogeneous object detection using YOLOv3. The system successfully establishes client–server communication, enables local training on Raspberry Pi clients, and executes model aggregation on the central server.

4.2 Results Obtained

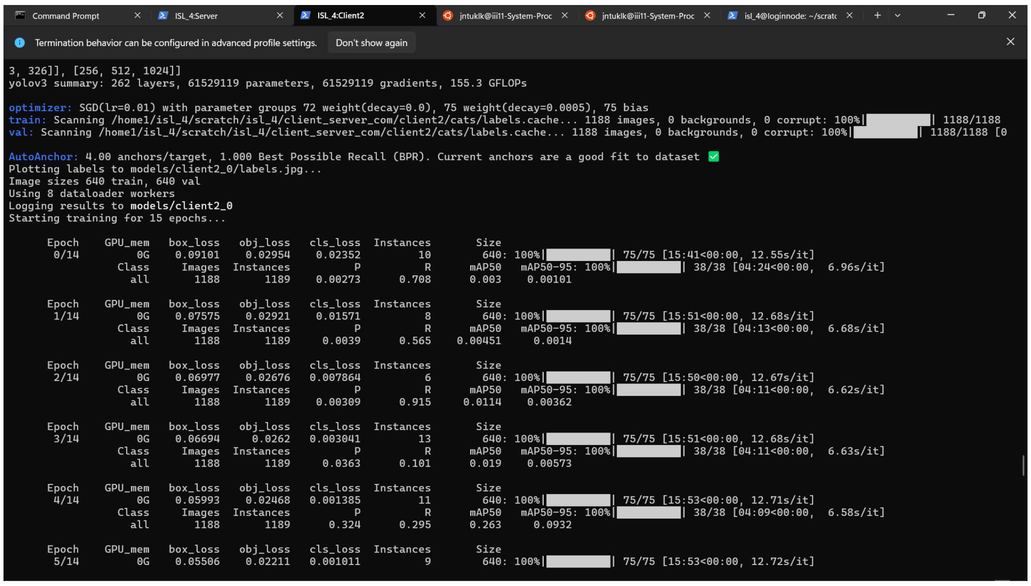
The following milestones were achieved during the implementation phase:

- Successful establishment of communication between Raspberry Pi clients and the central server.
- Independent local training of YOLOv3 on each Raspberry Pi using a partitioned dataset.
- Correct execution of global model aggregation using the FedAvg algorithm.
- Verification of aggregated model redistribution to all clients.



```
(pytorch_gpu) isl_4@loginnode:~/scratch/isl_4/client_server_com/server$ vim server.py
(pytorch_gpu) isl_4@loginnode:~/scratch/isl_4/client_server_com/server$ python server.py
[PHASE 1] Server listening on 127.0.0.1:5000
[+] Connected by ('127.0.0.1', 35730)
[SERVER] Sending status: Checkpoint does not exist
[PHASE 1] Connection window closed. No more clients will be accepted.
[PHASE 3] Waiting for client model uploads to complete...
```

Figure 4.1: Federated learning setup and aggregation phase demonstration.


Command Prompt
ISL_4:Server
ISL_4:Client2
jntuklk@iii11-System-Proc
jntuklk@iii11-System-Proc
isl_4@loginnode: ~/scratc
Don't show again

Termination behavior can be configured in advanced profile settings.

```
3, 326]], [256, 512, 1024]]
yolov3 summary: 262 layers, 61529119 parameters, 61529119 gradients, 155.3 GFLOPs

optimizer: SGD(lr=0.01) with parameter groups 72 weight(decay=0.0), 75 weight(decay=0.0005), 75 bias
train: Scanning /home1/isl_4/scratch/isl_4/client_server_com/client2/cats/labels.cache... 1188 images, 0 backgrounds, 0 corrupt: 100%|██████████| 1188/1188
val: Scanning /home1/isl_4/scratch/isl_4/client_server_com/client2/cats/labels.cache... 1188 images, 0 backgrounds, 0 corrupt: 100%|██████████| 1188/1188 [0

AutoAnchor: 4.00 anchors/target, 1.000 Best Possible Recall (BPR). Current anchors are a good fit to dataset ✅
Plotting labels to models/client2_0/labels.jpg...
Image sizes 640 train, 640 val
Using 8 dataloader workers
Logging results to models/client2_0
Starting training for 15 epochs...

      Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
       0/14         0G    0.09101    0.02954    0.02352         10        640: 100%|██████████| 75/75 [15:41<00:00, 12.55s/it]
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100%|██████████| 38/38 [04:24<00:00,  6.96s/it]
                   all       1188       1189    0.00273      0.708      0.003    0.00101

      Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
       1/14         0G    0.07575    0.02921    0.01571          8        640: 100%|██████████| 75/75 [15:51<00:00, 12.68s/it]
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100%|██████████| 38/38 [04:13<00:00,  6.68s/it]
                   all       1188       1189     0.0039      0.565    0.00451     0.0014

      Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
       2/14         0G    0.06977    0.02676   0.007864          6        640: 100%|██████████| 75/75 [15:50<00:00, 12.67s/it]
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100%|██████████| 38/38 [04:11<00:00,  6.62s/it]
                   all       1188       1189    0.00309      0.915     0.0114    0.00362

      Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
       3/14         0G    0.06694     0.0262   0.003041         13        640: 100%|██████████| 75/75 [15:51<00:00, 12.68s/it]
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100%|██████████| 38/38 [04:11<00:00,  6.63s/it]
                   all       1188       1189     0.0363      0.101      0.019    0.00573

      Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
       4/14         0G    0.05993    0.02468   0.001385         11        640: 100%|██████████| 75/75 [15:53<00:00, 12.71s/it]
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100%|██████████| 38/38 [04:09<00:00,  6.58s/it]
                   all       1188       1189      0.324      0.295      0.263     0.0932

      Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
       5/14         0G    0.05506    0.02211   0.001011          9        640: 100%|██████████| 75/75 [15:53<00:00, 12.72s/it]
```

Figure 4.2: Sample training progress on simulated clients (placeholder figure).

These initial results validate the functional integration of federated learning components within a heterogeneous environment. The communication protocol, synchronization, and aggregation mechanisms were all verified through experimental testing.

4.2.1 Performance Observations

Although full quantitative evaluation is planned for later stages, several qualitative observations were noted:

- Local YOLOv3 training runs successfully on Raspberry Pi devices for smaller batch sizes and shorter epochs.
- The FedAvg aggregation correctly combines model parameters when compatible weight structures are maintained.
- Communication overhead remains manageable within a local Wi-Fi network.

4.3 Challenges Faced

During experimentation, a few technical challenges emerged that limited the end-to-end performance of the federated setup:

4.3.1 Parameter Mismatch During Aggregation

The YOLOv3 model trained independently on each Raspberry Pi occasionally produced slightly inconsistent parameter shapes or key names in the state dictionaries. When these were aggregated at the server side, the model sometimes failed to load correctly or produce valid predictions. Future work will involve developing parameter alignment or layer-mapping techniques to ensure compatibility before aggregation.

4.3.2 Hardware Limitations

The limited computational and memory capacity of Raspberry Pi devices constrained the batch size and training duration. Extended training sessions often resulted in thermal throttling and slower epoch completion. Despite these constraints, the prototype successfully demonstrated the feasibility of federated object detection on edge hardware.

4.3.3 Network Stability

Occasional packet drops and Wi-Fi latency caused interruptions in model weight transfers between clients and the server. While not critical for small-scale experiments, such issues may significantly affect scalability and performance in large deployments.

4.4 Discussion

The successful execution of federated training and aggregation confirms that heterogeneous devices can collaboratively train a deep neural network without centralized data collection. However, further work is required to improve model consistency, optimize data partitioning, and implement more sophisticated federated optimization algorithms such as FedProx or FedNova to handle system heterogeneity.

4.5 Summary

This chapter discussed the experimental results and challenges observed in the federated object detection system. The implementation confirmed that federated learning can be realized across devices with varying computational capacities. The following chapter presents the summary of findings and outlines future directions for enhancing the framework.

CHAPTER 5

SUMMARY AND CONCLUSION

This project investigated the feasibility, challenges, and performance characteristics of deploying Federated Learning (FL) for heterogeneous object detection using the YOLOv3 architecture across resource-constrained edge devices. The work focused on designing a functional FL framework from first principles, implementing it on real hardware (two Raspberry Pi clients and a central laptop server), and analyzing system behaviour through experimentation.

5.1 Summary of Work

The thesis began by identifying the key motivation for this study: traditional centralized training of object detection models raises privacy, bandwidth, and data-ownership concerns, especially in distributed IoT environments. Federated Learning offers a decentralized alternative where clients collaborate without sharing raw data.

Chapter 1 introduced the problem and outlined the objectives, including building a custom FL system, integrating YOLOv3, evaluating cross-device performance, and identifying challenges related to heterogeneous edge hardware.

Chapter 2 provided an extensive review of relevant literature, covering FL fundamentals, communication-efficient algorithms such as FDA and FDA-OPT, hierarchical aggregation strategies, and model-contrastive approaches like MOON. It also reviewed YOLO-based object detection, highlighting why YOLOv3 was chosen for edge deployment. The chapter concluded by presenting the system architecture used in this work.

Chapter 3 detailed the methodology, including hardware/software configuration, communication protocols, and the mathematical foundation of the FedAvg algorithm. A major contribution of this chapter was the exploration of alternative implementations—Flower FL, zero-shot learning approaches, and TensorFlow re-implementation—to investigate root causes behind observed model inconsistencies. These attempts confirmed that the architectural mismatch originated from YOLOv3 itself rather than the custom FL setup.

Chapter 4 presented experimental results showing successful communication, local training on Raspberry Pi clients, and global aggregation on the server. Despite these successes, the aggregated YOLOv3 model failed to perform object detection due to parameter misalignment. Additional challenges such as limited computational capacity and network instability further impacted performance.

5.2 Key Findings

The major findings of the project are summarized below:

- **Federated Learning is feasible on heterogeneous edge devices.** The custom FL pipeline demonstrated successful communication, local training, and aggregation across Raspberry Pis and a laptop server.
- **YOLOv3 is highly sensitive to decentralized training.** Small divergences during local updates resulted in inconsistent parameter structures, making the aggregated model unusable.
- **The mismatch is not caused by the implementation.** Replicating the setup using the Flower FL framework reproduced the exact same issues, confirming the problem is tied to YOLOv3's architecture.

5.3 Conclusion

This work successfully demonstrated the fundamental components of a federated learning framework for object detection across heterogeneous devices. Although a fully functional aggregated YOLOv3 model was not achieved, the experiments provided critical insight into architectural challenges associated with applying FL to complex object detection models.

The findings show that practical FL for object detection requires either:

- architectures inherently robust to heterogeneous updates, or
- additional parameter alignment and consistency mechanisms.

Despite limitations, the study confirms that Raspberry Pis can participate effectively in federated learning pipelines, provided the models are lightweight and synchronization is carefully managed.

5.4 Future Work

Based on the insights from this study, several promising research directions emerge:

1. **Exploring lightweight and FL-friendly object detection architectures** Models such as YOLOv5-Nano, MobileNet-SSD, or lightweight DETR variants may aggregate more consistently.

2. **Implementing advanced FL algorithms** Methods such as FedProx, FedNova, MOON, or FDA-OPT can reduce model drift and handle non-IID training more effectively.

3. **Parameter alignment or layer-freezing strategies** Freezing backbone layers or enforcing synchronized checkpoints may prevent architecture divergence.

4. **Model compression and quantization** Techniques such as pruning or 8-bit quantization can reduce compute demands on Raspberry Pis.

5. **Hierarchical or cluster-based aggregation** Inspired by AHA frameworks, grouping similar clients may result in more stable training.

6. **Using GPU-accelerated edge devices** Platforms like NVIDIA Jetson could stabilize training for heavier models like YOLOv3.

Final Remarks

Although the aggregated YOLOv3 model did not yield successful inference, this work establishes a strong foundation for further research in federated object detection. The insights gained from the challenges, experiments, and alternative approaches provide a valuable roadmap for future development of privacy-preserving distributed vision systems suited for real-world IoT and smart-city environments.

REFERENCES

[1] FDA-OPT: Communication-Efficient Federated Fine-Tuning of Language Models

[2] Model-Contrastive Federated Learning

[3] FEDERATED LEARNING FOR MOBILE KEYBOARD PREDICTION

[4] Real-World Image Datasets for Federated Learning

[5] Federated Object Detection for Quality Inspection in Shared Production