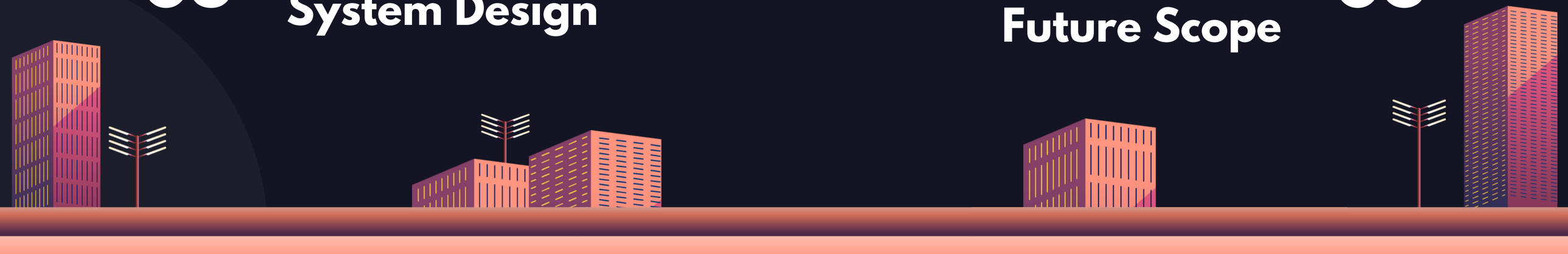
Containerized Federated Learning for Autonomous Driving

Nikita Miller
Aariz Faridi
Anisha Katiyar
Yatish Sikka

Table of Contents



01
Introduction

Motivation

In real-world applications, sharing raw data across institutions or devices is often infeasible due to privacy, bandwidth, or regulatory constraints.

Federated Learning (FL) enables collaborative model training without sharing local data.

However, deploying FL in practical settings is challenging due to system variability, scalability, and reproducibility. Therefore, we use MLOps to implement our solution.

Problem Statement

Training autonomous driving models centrally raises privacy, scalability, and reproducibility issues. Existing FL solutions lack environment consistency, model monitoring, and real-world deployment readiness. There's a need for a scalable, privacy-preserving pipeline with automated training and real-time performance monitoring.

Project Overview

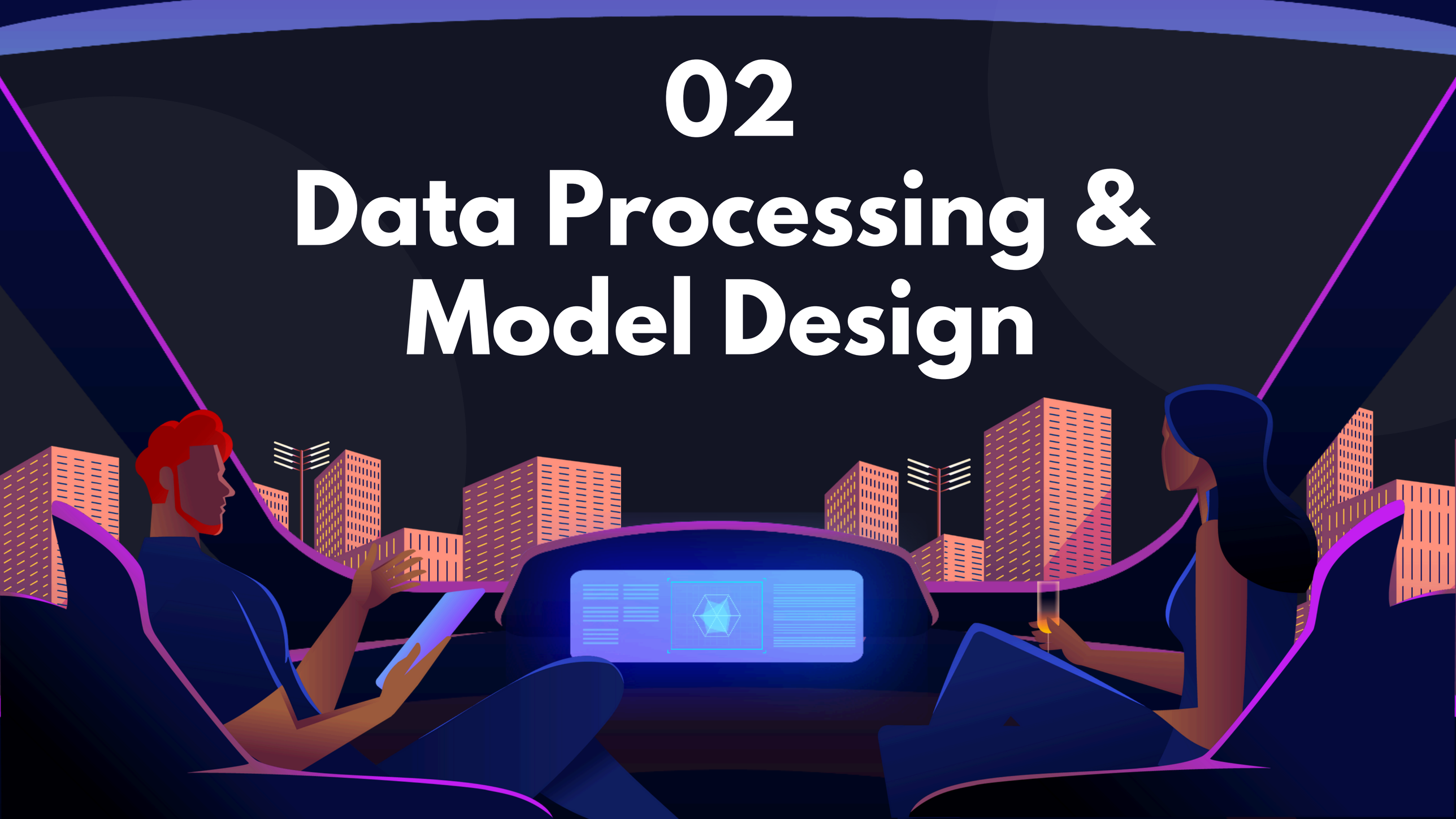
We integrated federated learning with containerization and MLOps to streamline deployment, monitoring, and continuous model updates. Vehicles act as client nodes, training YOLOv9 models on on-device data, while a central server securely aggregates model updates. This setup avoids raw data transmission and enables seamless collaboration across distributed nodes.

Key components include

- Flower for FL orchestration with secure FedAvg aggregation
- YOLOv9 for real-time object detection
- Docker for scalable deployment
- ClearML, AWS CloudWatch, and Optuna for performance monitoring and tuning

This modular and scalable architecture is tailored for real-world, privacy-sensitive deployment across autonomous vehicle environments.

02
Data Processing & Model Design

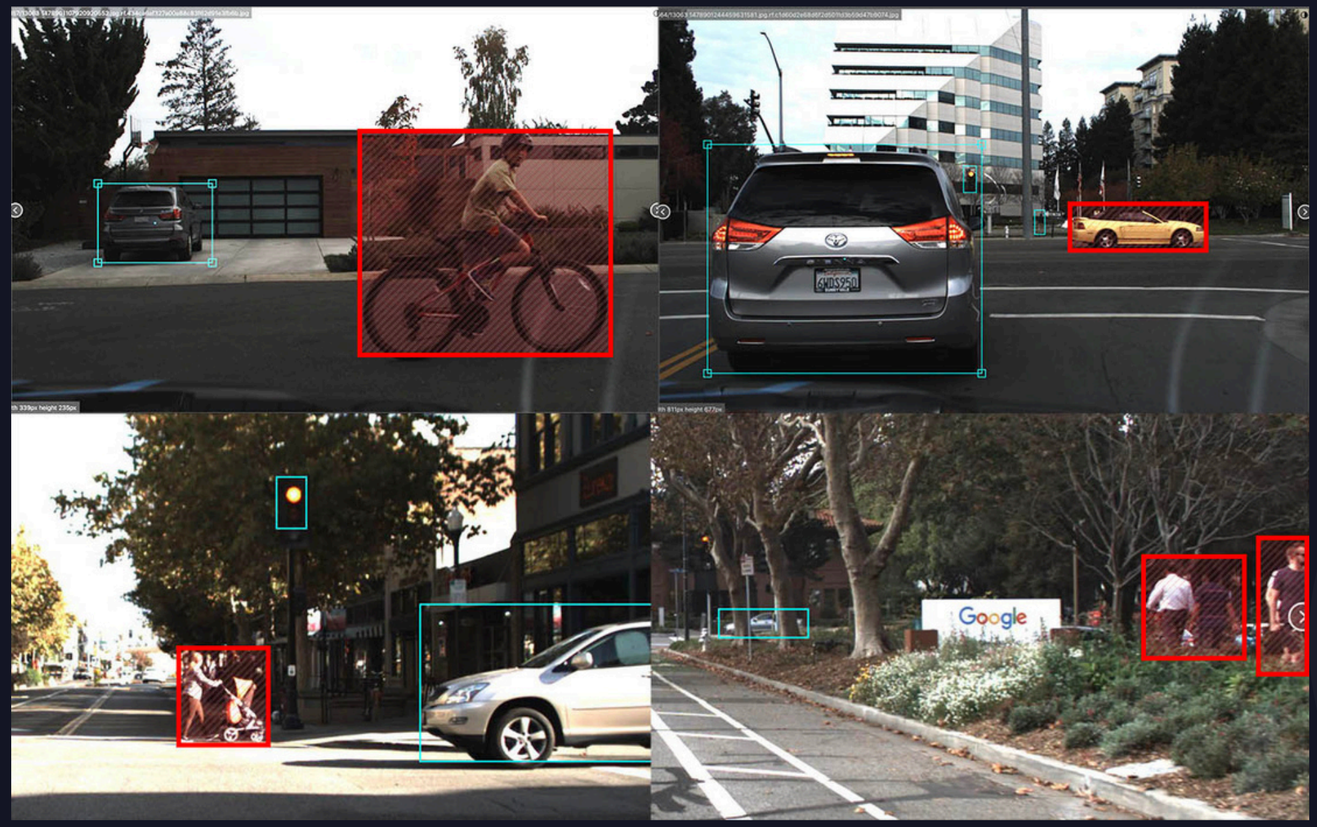


Dataset

The Udacity Self-Driving Dataset

Includes 15,000 images (1920×1200) with 97,942 hand-verified annotations across 11 object classes and 1,720 null examples

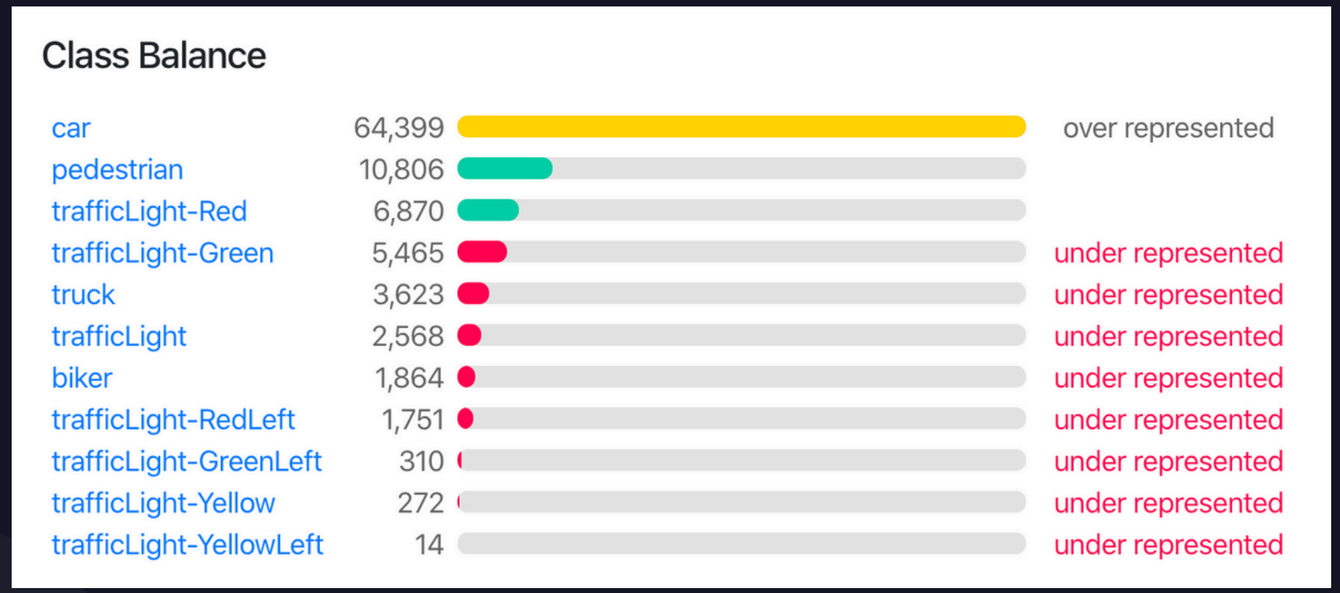
We used a downsampled 512×512 version (~580MB) suitable for YOLOv9t (tiny).



https://public.roboflow.com/object-detection/self-driving-car

Class Imbalance

Augmentations were applied only to underrepresented classes to avoid bias and were applied randomly



Techniques used:

- Horizontal flip
- Brightness/contrast adjustment
- Gaussian blur
- Random rotation & scaling

YOLOv9t Model Architecture

Real-time object detection model, ideal for edge devices like autonomous vehicles.

PGI (Programmable Gradient Information): Improves gradient flow during backpropagation.

GELAN (Generalized Efficient Layer Aggregation Network): Enhances feature reuse and multi-scale learning.

Fast inference suitable for real-time processing

High mAP across dense scenes

Lightweight architecture fits well in a federated setting

Federated Learning

Definition

Decentralized ML approach where models are trained collaboratively across multiple devices or nodes without sharing raw data.

Only model updates like weights or gradients are exchanged with a central server for aggregation.

Server and client architecture

Benefits

Preserves privacy of sensor data (images, GPS, etc.) since it never leaves the client device

Reduces bandwidth usage by transmitting only model parameters and not large datasets

Enables real-time learning at the client side as models continuously improve while staying local

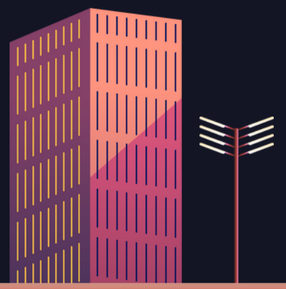
Usage

Vehicle/Clients train YOLOv9 models on their own local driving datasets

A central server running Flower aggregates updates using FedAvg

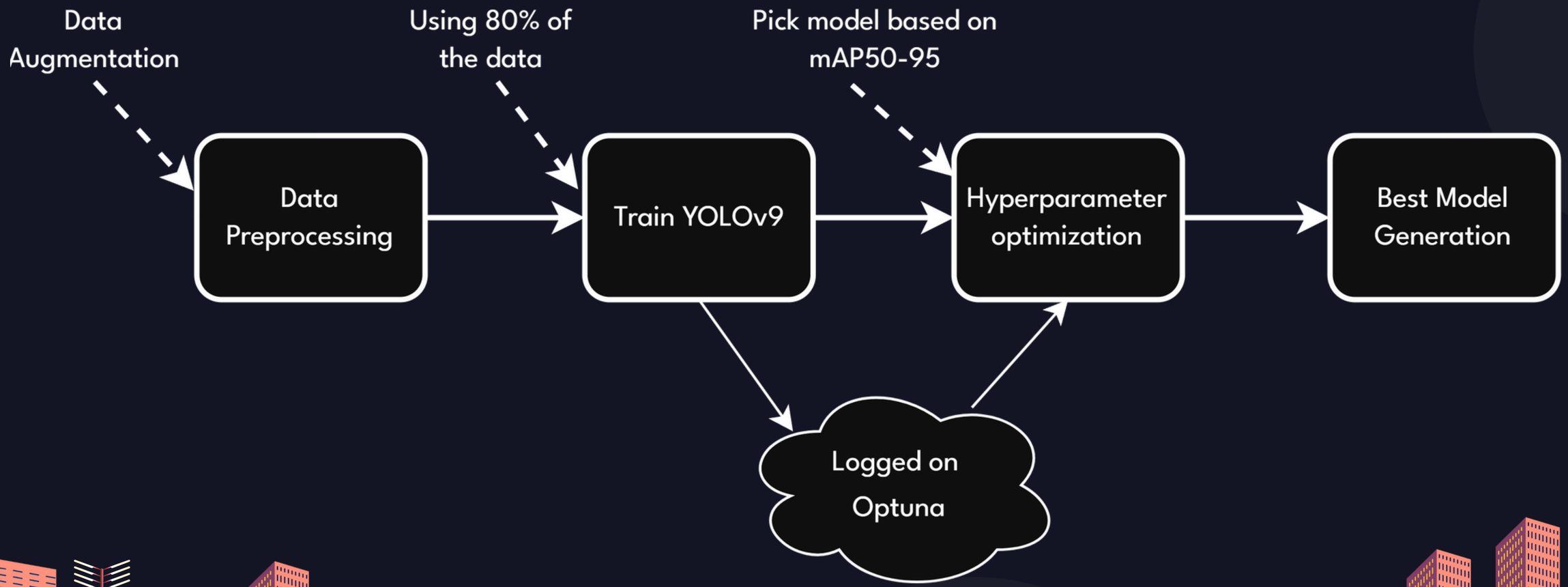
Updated global model is then redistributed to clients for the next training round

ClearML tracks performance across rounds



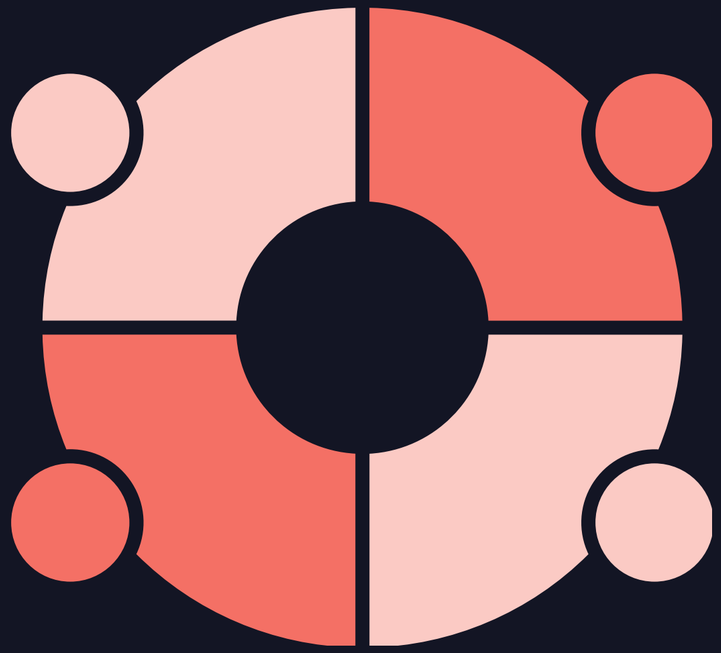
03
System Architecture & Tools

Flowchart for Model Generation



Hyperparameter Tuning

Optuna is an automatic hyperparameter optimization framework.



It uses Bayesian optimization to predict the best set of parameters using past trials rather than brute force.

Focuses on best regions, avoids wasteful trials unlike Grid/Random Search.

Supports pruning of bad runs early so faster convergence

Hyperparameters Used

- Learning Rate (lr0): 1e-5 to 5e-3 (log scale)
  - A good learning rate is critical — too low slows convergence, too high destabilizes training.

- Momentum: 0.85 to 0.97
  - Controls how much past gradients influence the current step. Lower = responsive, higher = smoother updates.

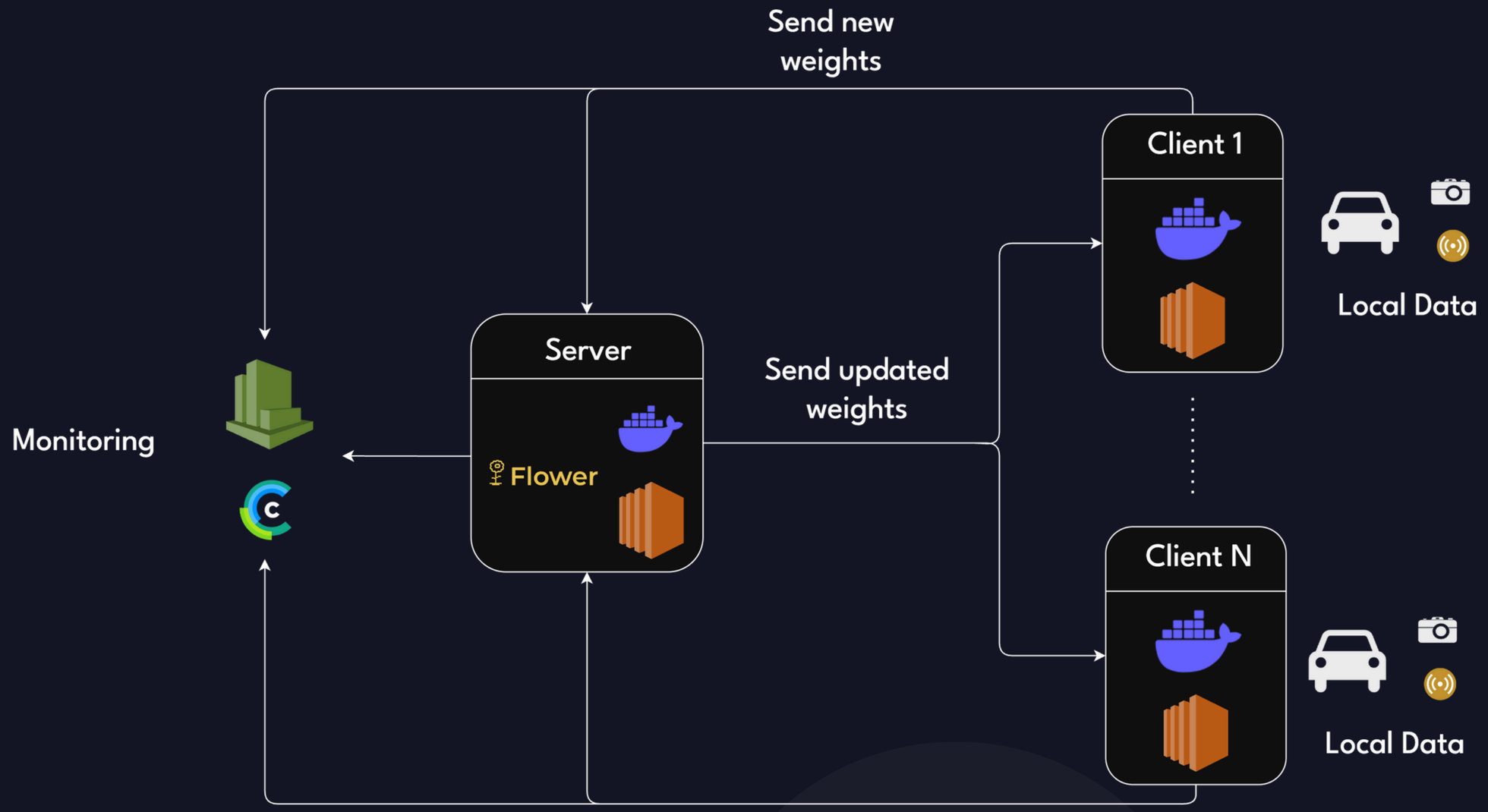
- Weight Decay: 1e-6 to 1e-3
  - Regularizes weights to prevent overfitting.

- Warmup Epochs: 1 to 5
  - Gradually increases learning rate early on to avoid large, unstable updates.

Hyperparameters Used

- **HSV Hue Shift (hsv_h): 0.0 to 0.05**
  - Controls color jittering in data augmentation.
- **Mosaic Augmentation: 0.5 to 1.0**
  - Combines 4 images into 1 for better generalization.
- **MixUp Augmentation: 0.0 to 0.3**
  - Blends images and labels, high values may over-blur training.
- **Rotation (degrees): 0 to 10**
  - Simulates rotated objects, larger values distort too much.
- **Scale Jitter (scale): 0.4 to 1.0**
  - Applies zoom-in/out augmentation.

System Architecture



Docker

Containerized YOLOv9 FL Client

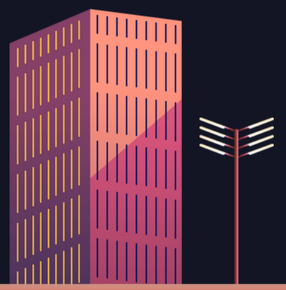
We built a custom Docker image encapsulating YOLOv9, Flower, ClearML, and all dependencies for a consistent, isolated client training environment.

Version-Controlled Deployment via Docker Hub

The Docker image was pushed to Docker Hub, enabling easy versioning and centralized access across distributed infrastructure.

Seamless EC2 Integration

On launching an EC2 instance, we pulled the Docker image directly from Docker Hub, ensuring rapid, reproducible client setup on AWS instances.



AMI for Scalability

AMI Snapshot of Pre-Configured EC2 Client

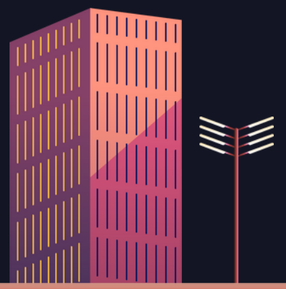
After configuring the EC2 instance with Docker and our federated client setup, we created a custom Amazon Machine Image (AMI) to capture this state.

Instant Scalability for FL Clients:

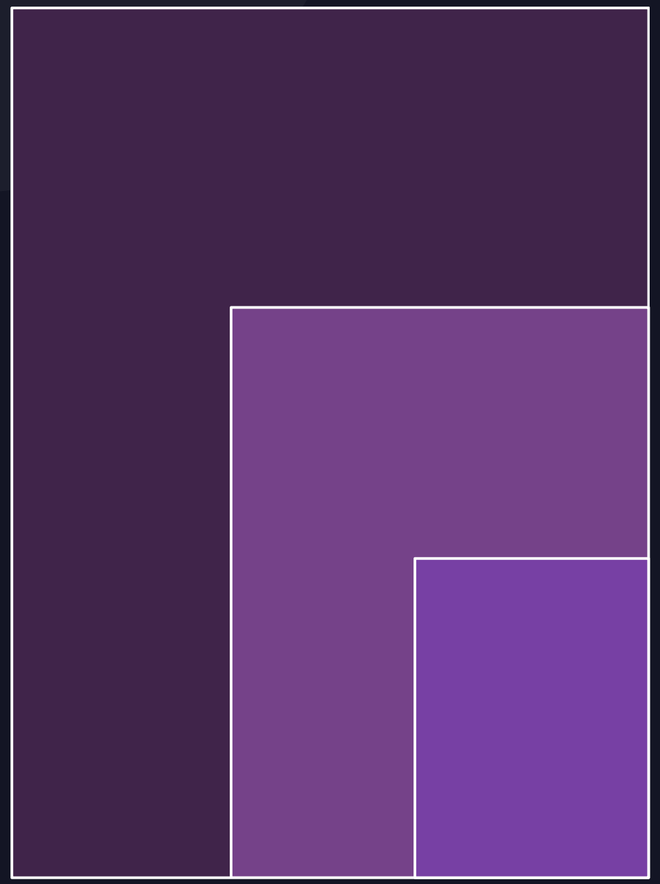
Using this AMI, we can rapidly spin up new EC2 instances pre-loaded with the entire FL client stack, reducing provisioning time to under a minute.

Automation-Ready for Auto-Scaling & Recovery:

The AMI serves as the foundation for Auto Scaling Groups, enabling one-click client instantiation, seamless crash recovery, and dynamic scaling during FL rounds.



Monitoring and Drift Detection



Used ClearML to monitor:

- Server average mAP50-95 scores.
- Drift detection (>10% drop in mAP score triggers re-initialization)
- FL round time

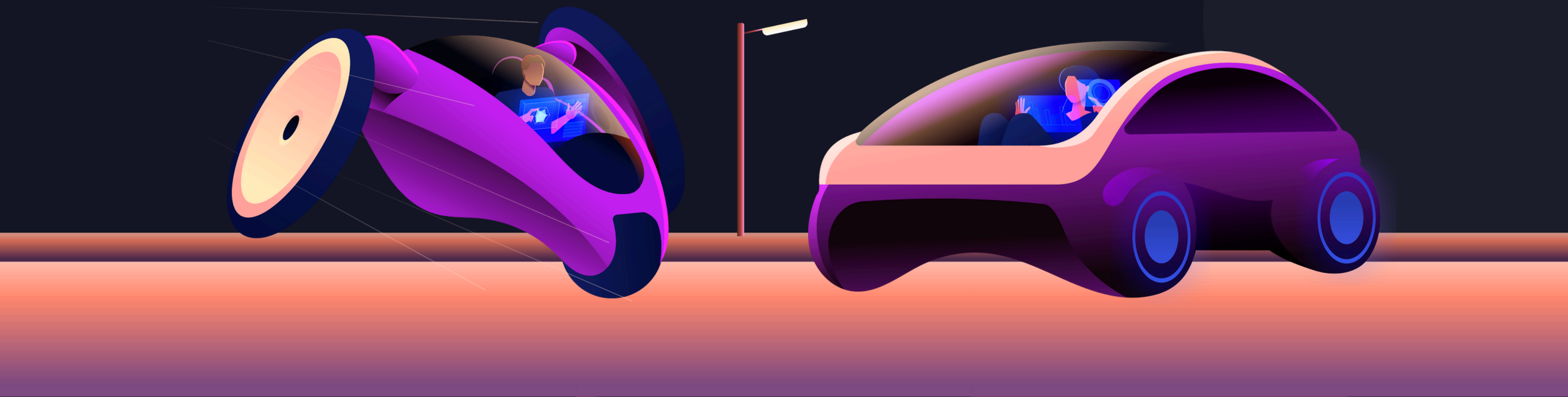
AWS CloudWatch used to log:

- CPU Active Usage
- Memory Usage
- Net bytes sent/received

Tools Summary

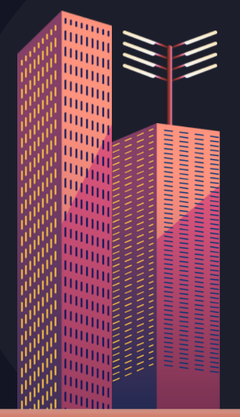
Tool	Category	Purpose
Optuna	MLOps	Performed hyperparameter tuning for YOLOv9 baseline
Flower	Federated Learning	Orchestrates communication between clients and server and performs secure weight aggregation via FedAvg
Docker	Containerization	Containerized each client node and server for reproducible and isolated execution
ClearML	MLOps	Tracked model metrics (mAP50–95), model drift, system metrics, client participation and FL round duration
AWS CloudWatch	MLOps	Monitored system usage and infrastructure-level logs

04
Demo



Live Demo

We will create a new client using an Amazon Machine Image (AMI) and connect it to the dynamic scaling enabled FL server.



05 Evaluation and Performance Metrics

Metrics

mAP@0.5

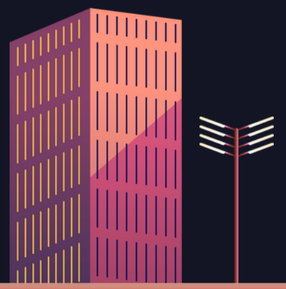
Measures precision at a 50% IoU threshold — helps evaluate how accurately the model detects object boundaries.

mAP@0.5-0.95

A more rigorous metric averaged over multiple IoU thresholds (0.5 to 0.95), providing a holistic view of detection quality and robustness.

Scalability

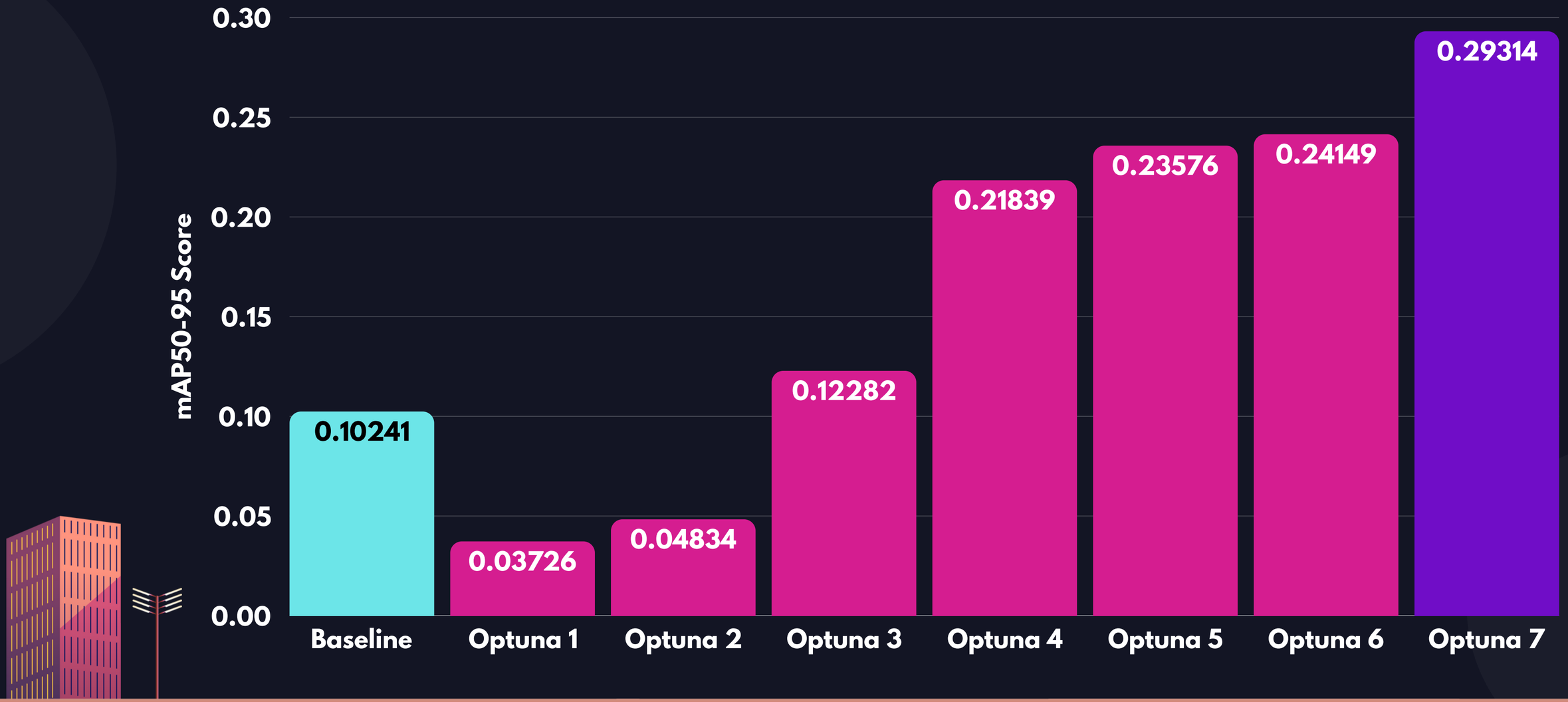
Assesses how effectively the system can instantiate new clients and maintain performance as the number of federated nodes increases.



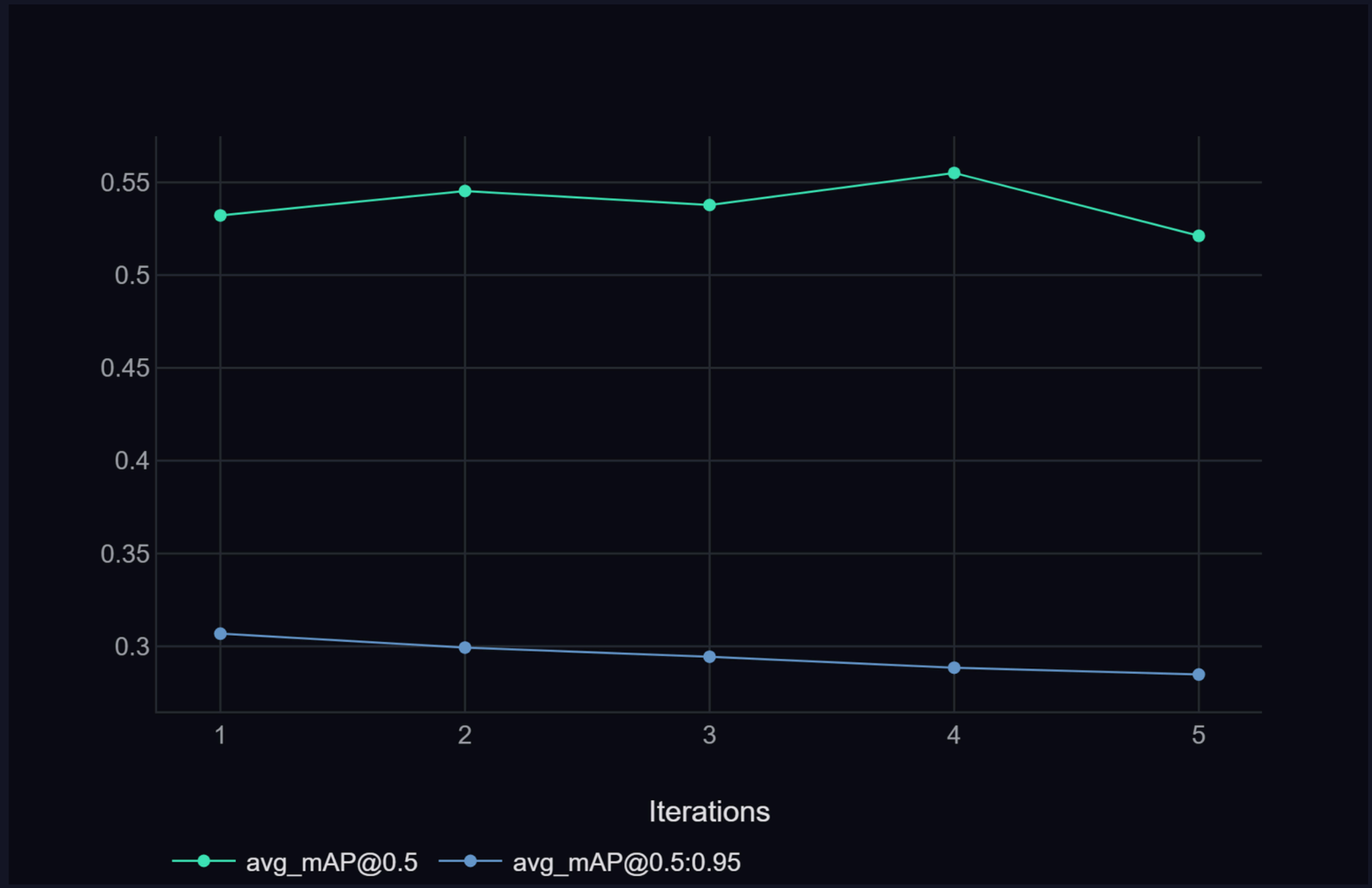
mAP : mean Average Precision

IoU : Intersection over Union

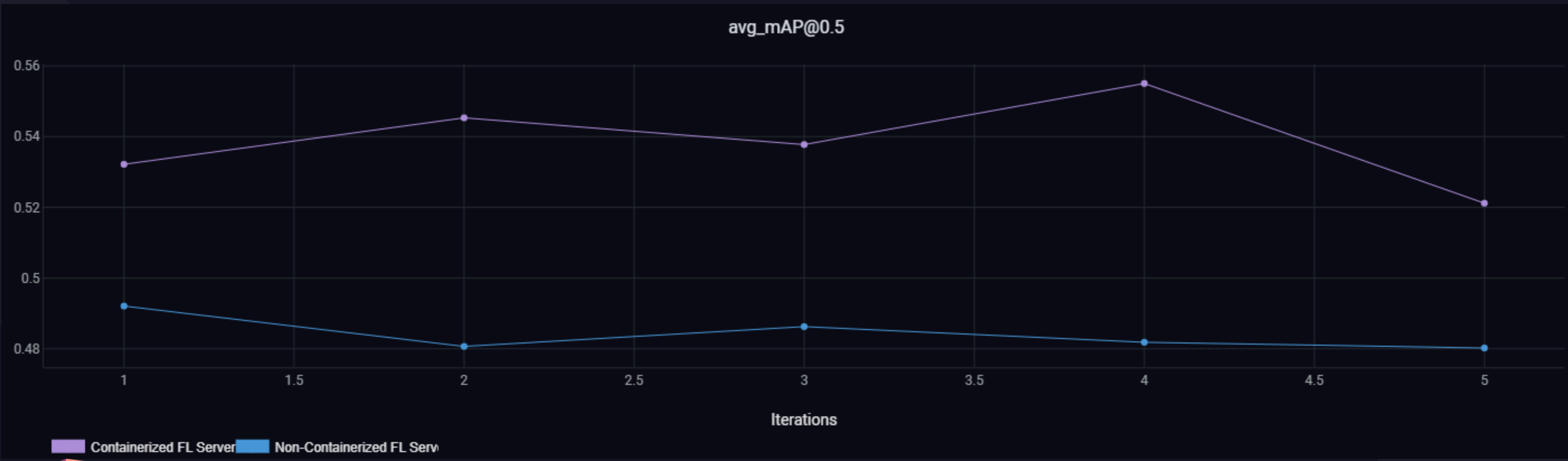
Hyperparameter Optimization



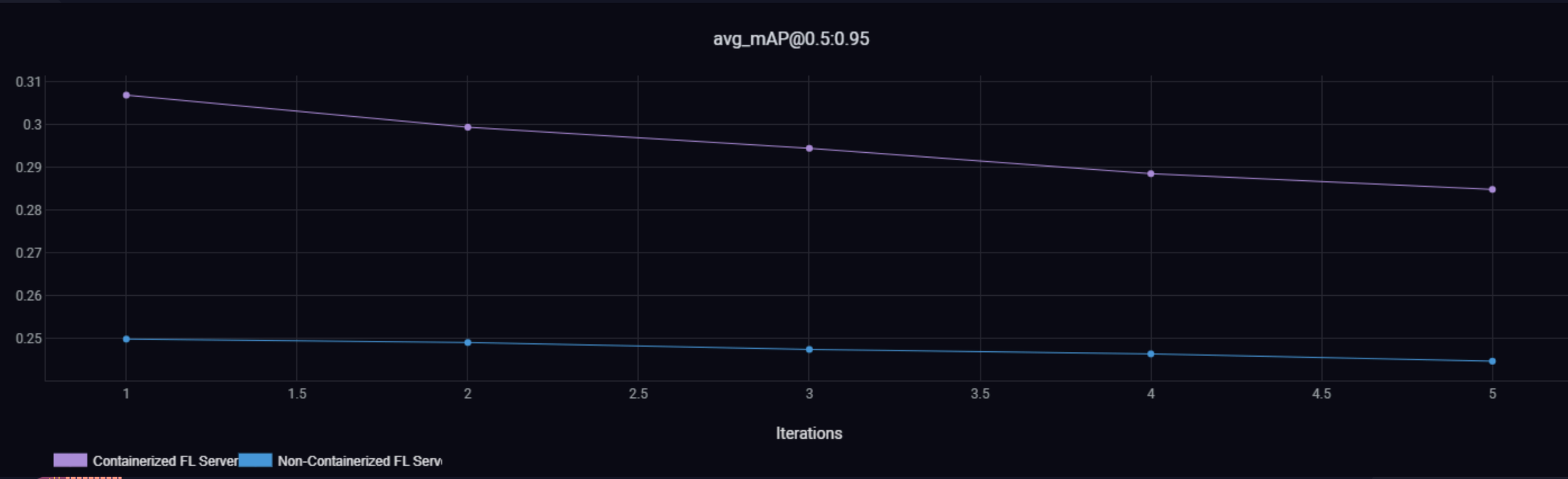
Drift Detection



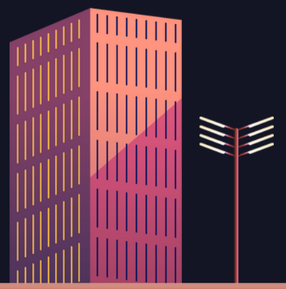
Model Performance



Model Performance



Scalability Performance

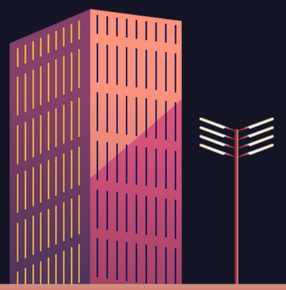


Why performance improved ?

Containerization introduced a stable environment

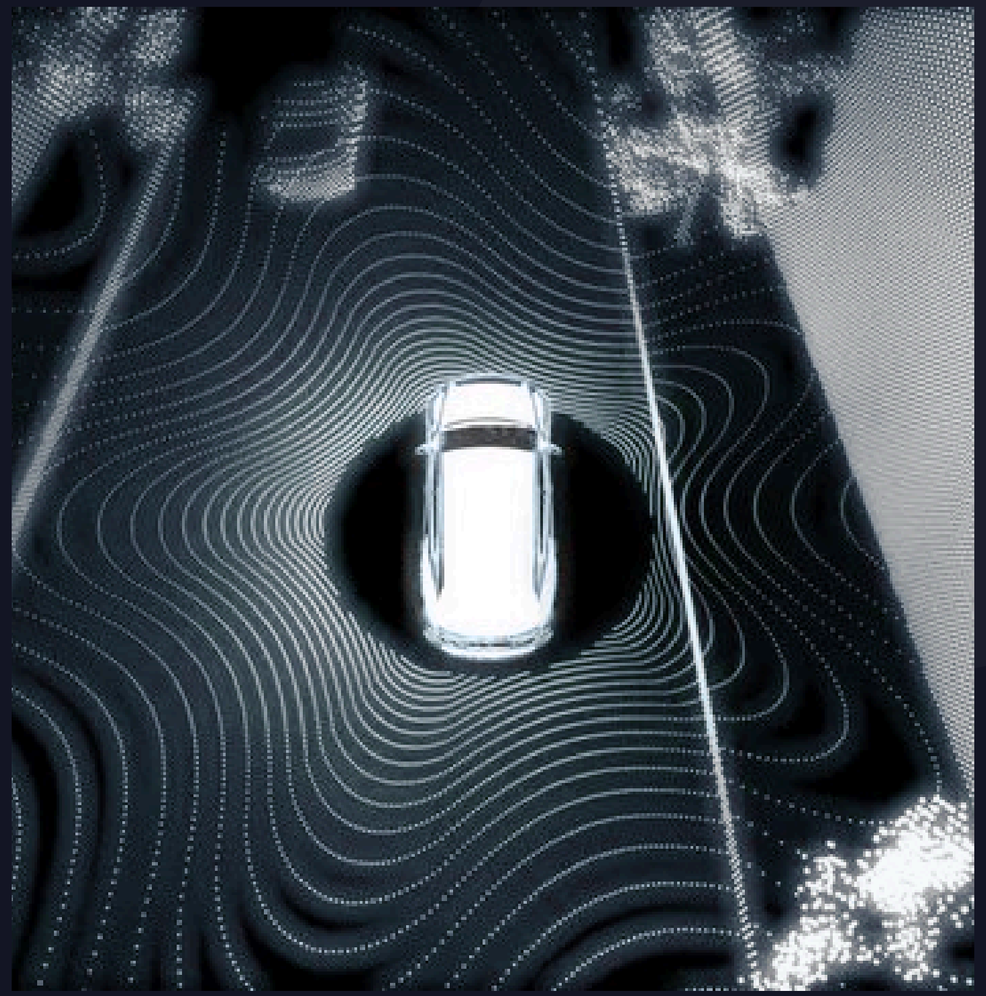
Dependency consistency

Controlled resource allocation



06

Limitations & Future Scope



Limitations

Simulated Clients

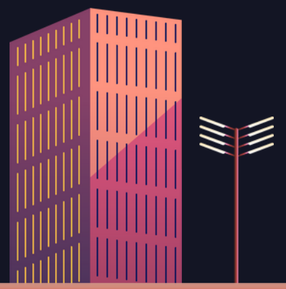
FL clients were run in isolated containers on cloud infrastructure, not deployed to actual vehicles or edge devices.

Limited Number of FL Rounds

Evaluation was done over 5 FL rounds due to time and compute constraints while full convergence may need more.

Fixed Non-IID Data Split

Each client had a static subset of data and real-world deployments may require dynamic data distribution handling.



Future Scope

Deploy on Real Edge Clients

Test performance of YOLOv9 inference and FL synchronization on embedded systems like Jetson Nano, Raspberry Pi.

Use various different Sensors

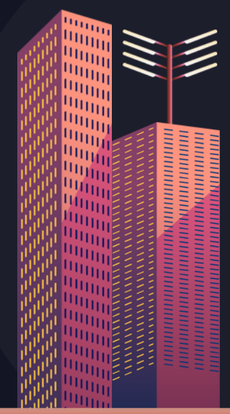
Integrate LiDAR, GPS, and temporal context to move toward multi-modal FL for autonomous navigation.

Deploy via Container Orchestration (ECS/Kubernetes)

Automate client/server scaling, fault tolerance, and load balancing using ECS or EKS to manage a large FL ecosystem.

Dynamic Client Participation and Failure Handling

Handle random dropouts, asynchronous training, and stale model updates effectively.



Thank You!