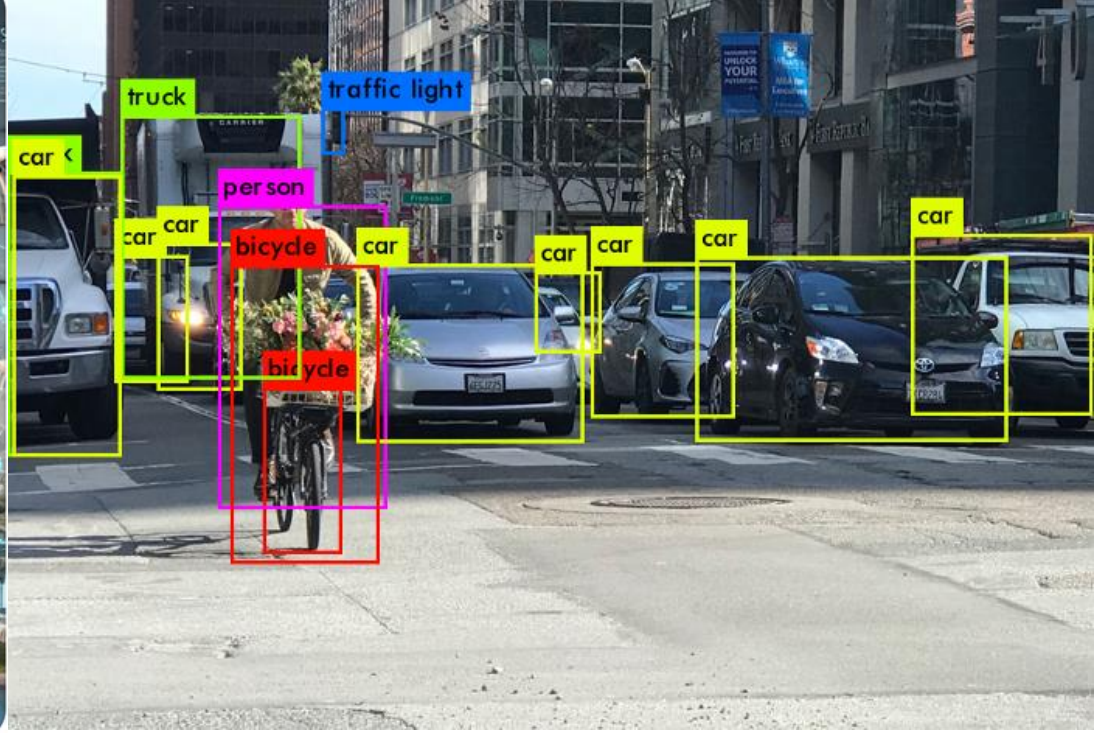


- **PROJECT TITLE:** AI-ENABLED DRONE SWARM NETWORK AND INFORMATION MANAGEMENT
  SUBTITLE: OBJECT DETECTION, DRONE2DRONE INFORMATION EXCHANGE AND COLLABORATIVE AWARENESS

SWARM NETWORKS – NEW GENERATION NETWORK MANAGEMENT

AI-BASED DRONE2DRONE INFORMATION MANAGEMENT

PROJECT – OPERATIONAL CONDITIONS

Students will be engaged to groups dedicated to work on tasks

1. Students will be split in 4 groups depending on the tasks they will be working on
2. The groups will have specific target outcomes

Students will be responsible to deliver their own tasks and contribute to the integration with other groups

The level of completeness and accuracy of task outcome, as well as dedication to the task will determine the grade of the student effort

Every Tuesday all groups will have meetings during the course to present the status of work progress

Fridays will be used for system integration meetings and targeted presentations

TEAM COLLABORATION AND ROADMAP

Students will be responsible to provide the integration of the solutions

- Integration & Evaluation : Ensure seamless interoperability between all components.

Team Assignments

AI & Object Detection Team: Implement and optimize YOLOv5 / v8 for drone applications.

Communication Protocol Team: Develop and test the MQTT-based data exchange framework.

LLM & Decision-Making Team: Fine-tune and deploy LLaMA2 for inter-drone understanding.(OR Rule-Based Communication (Replace LLMs with structured rule-based messaging))

Simulation & Testing Team: Set up ROS2 & Gazebo for multi-drone environments.

OBJECTIVE: DESIGN AND IMPLEMENT AN AI-BASED INTER-DRONE COMMUNICATION SYSTEM FOR COLLABORATIVE AWARENESS.

- **Phase 1: Individual module development (Weeks 1-8)**
  - Object detection implementation
  - MQTT setup for communication
  - LLaMA2 model training (OR **Rule-Based Communication** (Replace LLMs with structured rule-based messaging))
- **Phase 2: Initial system integration (Weeks 7-10)**
  - Connecting YOLOv5, MQTT, and LLaMA2
  - Testing information flow between drones
- **Phase 3: Simulation & Performance Evaluation (Weeks 10-12)**
  - Running full drone swarm simulations
  - Fine-tuning performance metrics
- **Phase 4: Final Testing & Report (Weeks 12-14)**
  - Refining all components
  - Documenting results and preparing presentations

OBJECTIVE: DESIGN AND IMPLEMENT AN AI-BASED INTER-DRONE COMMUNICATION SYSTEM FOR COLLABORATIVE AWARENESS.

Key Technologies:

- AI-based object detection (YOLOv5)
- MQTT for communication
- LLaMA2-based small LLM for inter-drone message interpretation (OR **Rule-Based Communication** (Replace LLMs with structured rule-based messaging))
- ROS2 & Gazebo for simulation

CHALLENGES IN DRONE SWARM COMMUNICATION

- High bandwidth consumption with raw sensor/image data exchange
- Real-time decision-making and obstacle avoidance
- Efficient collaborative awareness

PROPOSED SOLUTION

- Onboard AI for Local Processing (Detect and classify objects locally using YOLOv5)
- Optimized Information Exchange (Use MQTT for lightweight data sharing)
- LLM-Based Communication (Use LLaMA2 to enhance understanding and decision-making)

OBJECT DETECTION USING YOLOV5/V8

- **YOLOv5/v8 Overview:**
  - Fast, lightweight, and optimized for edge devices
  - Pre-trained on dataset, fine-tuned for drone environments
- **Implementation Workflow:**
  - Capture image from drone camera
  - Run inference using YOLOv5
  - Extract object type, position, and movement
  - Encode findings into structured data (JSON format)

OBJECT DETECTION USING YOLOV5

- **YOLOv5 Overview:**
  - Accessible via Google Colab or hosted API (Roboflow, Hugging Face)
  - Pre-trained on COCO dataset, fine-tuned for drone environments
- **Implementation Workflow:**
  - Upload image to YOLOv5 API
  - Run inference and receive detection results (object type, position)
  - Encode findings into structured JSON format

INFORMATION EXCHANGE USING MQTT

- **Why MQTT?**
  - Low bandwidth, lightweight, and ideal for real-time applications
  - Publish/Subscribe model for efficient communication
- **Implementation:**
  - Broker setup (Mosquitto)
  - Message format (JSON-encoded insights, not raw data)
  - Priority-based communication for critical alerts
  - Use Mosquitto MQTT (local or cloud-based)
  - Drones send detected objects and positions
  - Messages formatted as JSON (e.g., {"object": "car", "x": 5, "y": 10})

LLAMA2-BASED INTER-DRONE COMMUNICATION

- **Why Use LLMs?**
  - Enhance message comprehension and decision-making
  - Translate structured data into actionable commands
- **Fine-Tuning LLaMA2 for Drone Communication**
  - Dataset with typical drone interactions
  - Training using Hugging Face and PyTorch

SENSOR DATA PROCESSING

•**Objective:**Enable each drone to locally process sensor data (e.g., from LiDAR and cameras) to detect and recognize objects or environmental features. Instead of sharing raw data, drones should share extracted information.

•**Solution:**

- **Object Detection and Recognition:** Use YOLOv5 for real-time object detection. Models should be pre-trained on relevant datasets (e.g., COCO) and fine-tuned for specific environmental features relevant to the mission.
- **Data Summarization and Encoding:** Extract key information like object type, location, size, and movement vector.
  - Use compact JSON-like structures for encoding information.
  - Example:

```
{
  "object": "tree",
  "location": [34.5, -118.2, 15.0],
  "size": [2.5, 7.0],
  "velocity": [0, 0, 0]
}
```

SENSOR DATA PROCESSING

- **Implementation Tools:**
  - OpenCV for image processing.
  - PyTorch or TensorFlow for model deployment.

INFORMATION EXCHANGE PROTOCOL

Objective:

Facilitate reliable and low-latency communication between drones over a new generation network (e.g., 5G/6G).

Solution:

- **Communication Protocol Design:**
  - Use **MQTT** for lightweight publish-subscribe messaging.
  - Design custom message topics like /drone1/obstacles, /drone2/navigation, etc.
  - Implement priority-based message handling to ensure critical data (e.g., collision warnings) is delivered first.
- **Implementation Tools:**
  - **(ns-3** or **OMNeT++** for network simulation.)- not obligatory
  - **Paho MQTT** for messaging.

INTER-DRONE COMMUNICATION USING LLM

Objective:
Enable context-aware communication between drones using a small, fine-tuned LLM. The LLM should help interpret messages and facilitate collaborative decision-making.

Solution:

•Choosing the Model:

- Use a compact LLM such as **LLaMA 2 (7B variant)** suitable for edge inference.
- Fine-tune the model on a custom dataset with domain-specific dialogues, commands, and status reports.

•Message Encoding and Decoding:

- Encode messages with semantic meaning, e.g., translating numerical data into contextual statements.
- Example:

```
{
  "message": "Obstacle detected: Tree at 15m ahead. Suggested course adjustment: 5 degrees right."
}
```

INTER-DRONE COMMUNICATION USING LLM

Contextual Decision Making:

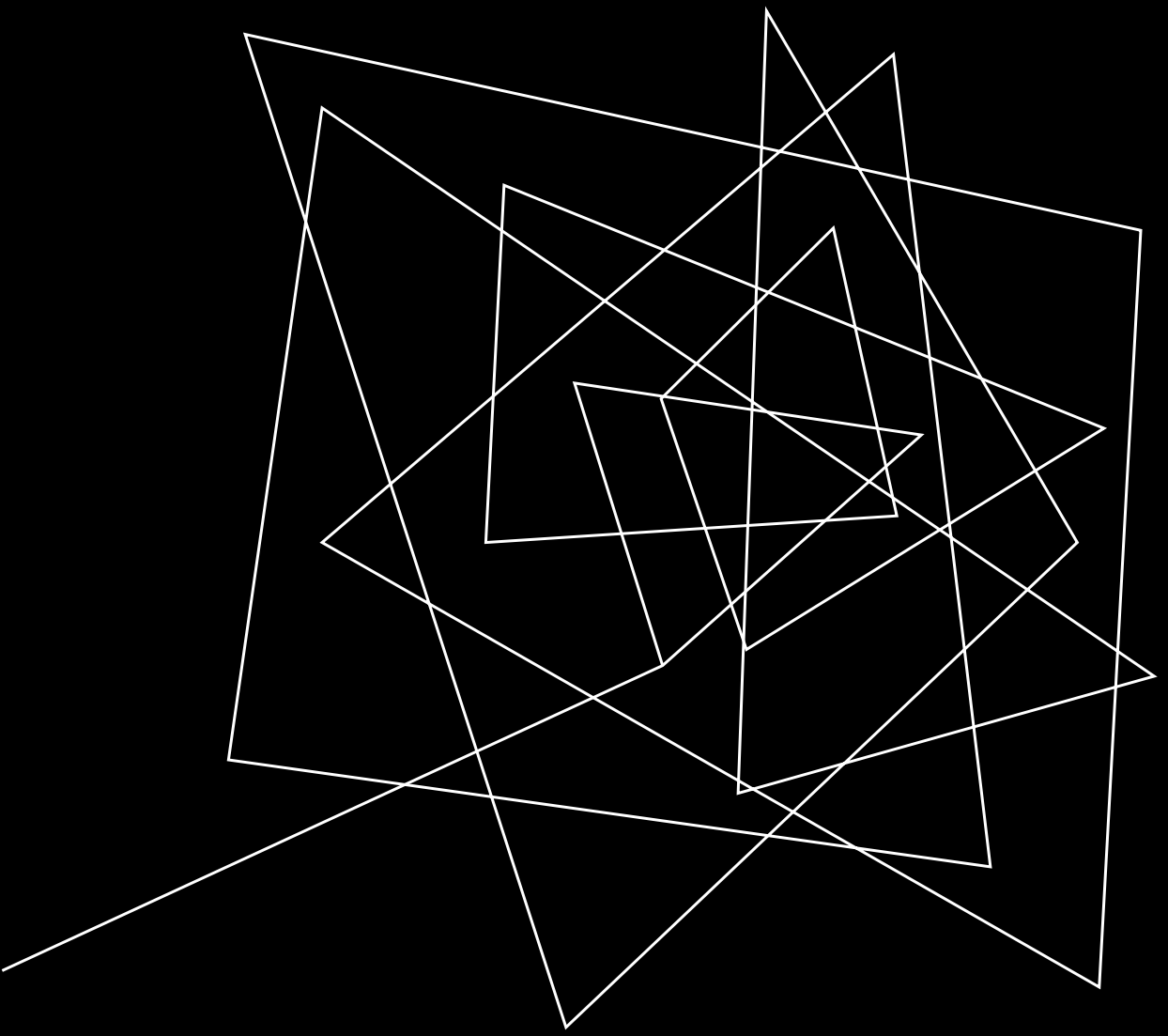
- The LLM should generate context-aware responses by understanding past exchanges and current mission goals.
- Implement conversation history management to maintain context across multiple messages.

Implementation Tools:

Hugging Face Transformers for model fine-tuning.

INTEGRATION AND TESTING

Simulation Environment:
Use ROS 2 with Gazebo for end-to-end testing.
Simulate environmental scenarios (e.g., forest, disaster zones).



IMPLEMENTATION

OBJECT DETECTION USING YOLOV8

Object detection

We will use the YOLOv5 model from the Ultralytics package for real-time object detection on drones.

Requirements:

ultralytics for YOLOv5

OpenCV for video processing

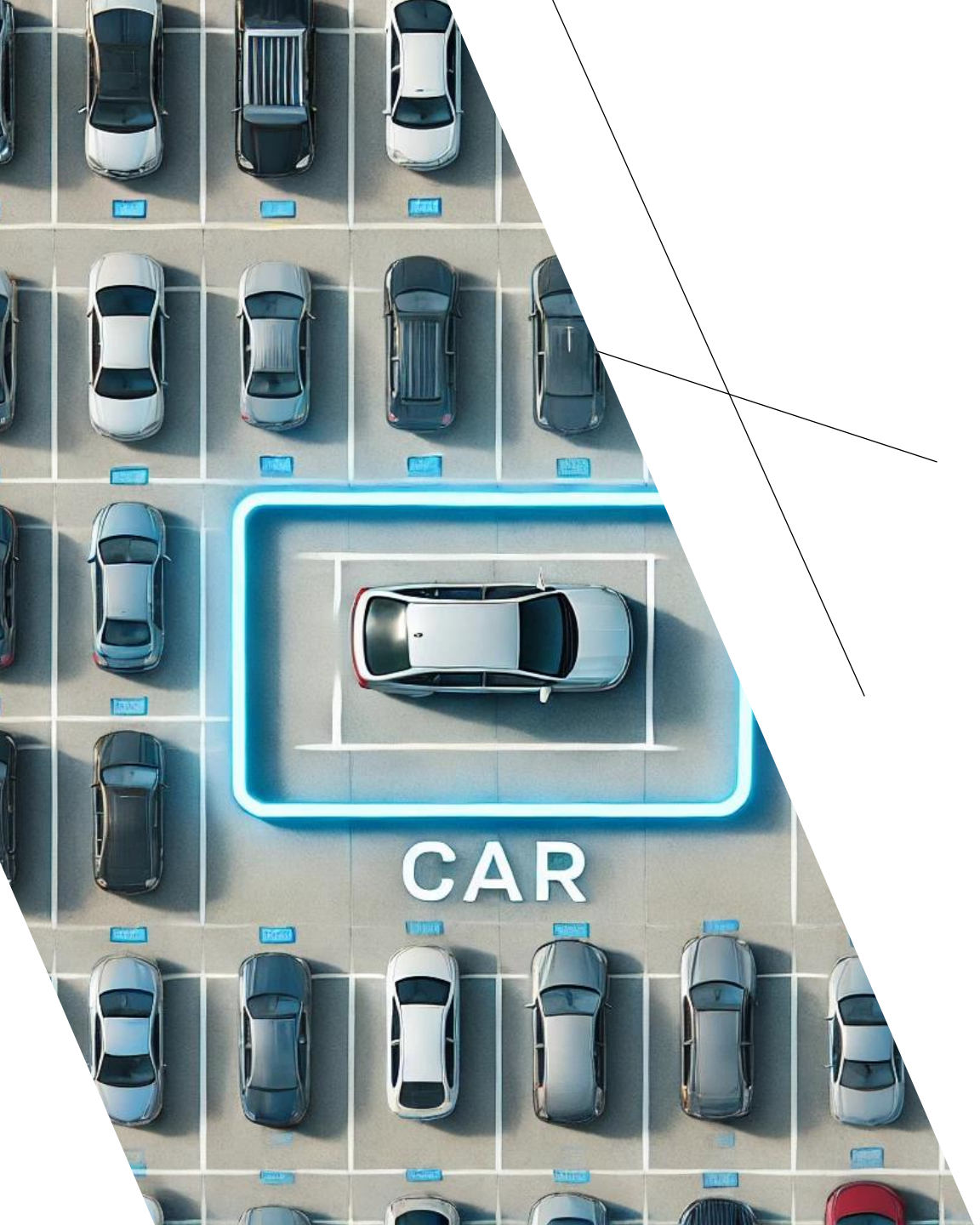
torch for PyTorch modeldeployment

Installation:

pip install ultralytics opencv-python-headless torch

CODE SAMPLE

```
import cv2

from ultralytics import YOLO


# Load YOLOv5 model

model = YOLO('yolov5n.pt')  # 'n' variant for lightweight and fast inference


# Open video stream (use 0 for webcam or provide video file path)

cap = cv2.VideoCapture(0)

while True:

    ret, frame = cap.read()

    if not ret:

        break

    # Run object detection

    results = model(frame)

    # Process detections

    for result in results.pred[0]:

        # Get bounding box, confidence and class

        x1, y1, x2, y2, conf, cls = result

        x1, y1, x2, y2 = map(int, [x1, y1, x2, y2])

        label = model.names[int(cls)]

        # Draw bounding box and label

        cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 255, 0), 2)

        cv2.putText(frame, f"{label} {conf:.2f}", (x1, y1 - 10),


cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)


    # Display the result

    cv2.imshow("YOLOv8 Object Detection", frame)

    if cv2.waitKey(1) & 0xFF == ord('q'):

        break

cap.release()

cv2.destroyAllWindows()
```



KEY FEATURES:

•Real-time object detection from a video stream or camera.

•Displays detected objects with bounding boxes and confidence scores.

Optimization for Edge Devices:

To optimize for NVIDIA Jetson or other edge devices, export the model to TensorRT:

yolo export model=yolov5n.pt format=engine

COMMUNICATION PROTOCOL DESIGN USING MQTT

This example demonstrates inter-drone communication using the MQTT protocol, enabling drones to exchange processed information efficiently.

Requirements :

paho-mqtt for MQTT communication

Install using:

pip install paho-mqtt

MQTT Broker Setup:

Use Eclipse Mosquitto as the MQTT broker.

Install it locally or use a cloud broker

To install Mosquitto on Ubuntu:

sudo apt update

sudo apt install mosquitto mosquitto-clients

sudo systemctl start mosquitto

sudo systemctl enable mosquitto

COMMUNICATION PROTOCOL DESIGN USING MQTT

Publisher (Drone 1):Drone 1 detects an obstacle and publishes processed information.

```
import paho.mqtt.client as mqtt
import json
import time


# MQTT Broker details
BROKER = 'localhost'
PORT = 1883
TOPIC = 'drones/obstacle'
# Data to publish
obstacle_data = {
    "drone_id": "drone_1",
    "object": "tree",
    "location": [34.5, -118.2, 15.0],
    "size": [2.5, 7.0],
    "velocity": [0, 0, 0]
}
```

```
# Data to publish
obstacle_data = {
    "drone_id": "drone_1",
    "object": "tree",
    "location": [34.5, -118.2, 15.0],
    "size": [2.5, 7.0],
    "velocity": [0, 0, 0]
def on_connect(client, userdata, flags, rc):
    print(f"Connected with result code {rc}")


# Setup MQTT Client
client = mqtt.Client()
client.on_connect = on_connect
client.connect(BROKER, PORT, 60)
```

```
# Publish information
while True:
    message = json.dumps(obstacle_data)
    client.publish(TOPIC, message)
    print(f"Published: {message}")
    time.sleep(5)  # Send every 5 seconds
```

COMMUNICATION PROTOCOL DESIGN USING MQTT

Subscriber (Drone 2):Drone 2 subscribes to the topic to receive obstacle information.

```
import paho.mqtt.client as mqtt
import json

# MQTT Broker details
BROKER = 'localhost'
PORT = 1883
TOPIC = 'drones/obstacle'

def on_connect(client, userdata, flags, rc):
    print(f"Connected with result code {rc}")
    client.subscribe(TOPIC)

def on_message(client, userdata, msg):
    data = json.loads(msg.payload)
    print(f"Received Data: {data}")
    # Process received data (e.g., adjust flight path)

# Setup MQTT Client
client = mqtt.Client()
client.on_connect = on_connect
client.on_message = on_message
client.connect(BROKER, PORT, 60)
client.loop_forever()
```

Key Features:

- Publisher-Subscriber Model: Ensures efficient, asynchronous communication.
- Lightweight Protocol: MQTT is suitable for bandwidth-constrained networks.

INTER-DRONE COMMUNICATION USING LLAMA2-BASED LLM

We’ll use a small variant of LLaMA2 (7B) fine-tuned for drone communication to interpret and contextualize messages..

Requirements:

transformers and accelerate from Hugging Face

Install using:

```
pip install torch transformers accelerate
```

Key Features:

Encodes messages with contextual awareness.

Generates responses that facilitate collaborative decision-making.

Code Sample:

```
from transformers import AutoTokenizer, AutoModelForCausalLM


# Load LLaMA2 Model and Tokenizer
model_name = "meta-llama/Llama-2-7b-hf"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForCausalLM.from_pretrained(model_name, device_map="auto")


# Encode message
message = "Obstacle detected: Tree at 15m ahead."
input_ids = tokenizer.encode(message, return_tensors='pt').to('cuda')


# Generate response
output = model.generate(input_ids, max_length=50)
response = tokenizer.decode(output[0], skip_special_tokens=True)


print("Drone Response:", response)
```

SIMULATION SETUP FOR DRONE SWARM

We will use **ROS 2** with **Gazebo** for simulating the drone swarm.

Requirements:

- ROS 2 Humble or Foxy
- Gazebo with MAVROS plugins

Key Components:

- Multiple drone models configured with LiDAR and camera sensors.
- Topics for inter-drone communication using ROS 2 nodes.
- Rviz for real-time visualization.

Simulation Steps:

1. Install ROS 2 and Gazebo.
2. Setup MAVROS for drone control.
3. Launch multiple drones in Gazebo with a shared ROS 2 namespace.
4. Use ROS 2 topics for inter-drone communication and information exchange.

AI MODEL TRAINING FOR INTER-DRONE INFORMATION EXCHANGE

Fine-tuning LLaMA2 on custom dialogues for inter-drone communication.

Dataset Preparation:

- Collect dialogues relevant to drone swarm communication, e.g., obstacle detection, navigation adjustments.
- Format data in **JSONL** with "input" and "output" pairs.

Fine-Tuning Script:

```
from transformers import AutoModelForCausalLM, AutoTokenizer, TrainingArguments, Trainer

model_name = "meta-llama/Llama-2-7b-hf"

tokenizer = AutoTokenizer.from_pretrained(model_name)

model = AutoModelForCausalLM.from_pretrained(model_name)
```

```
# Load dataset

from datasets import load_dataset

dataset = load_dataset('json', data_files={'train': 'train.jsonl', 'validation': 'valid.jsonl'})


# Training Configuration

training_args = TrainingArguments(

    output_dir="./llama2-drone",

    per_device_train_batch_size=2,

    per_device_eval_batch_size=2,

    evaluation_strategy="epoch",

    num_train_epochs=3

)
```

```
# Initialize Trainer

trainer = Trainer(

    model=model,

    args=training_args,


train_dataset=dataset['train'],


eval_dataset=dataset['validation']

)


trainer.train()
```

RUNNING YOLOV5 WITHOUT A DEDICATED GPU

- **Google Colab**: Run YOLOv5 in Google Colab, which provides free GPU access.
- **Remote Servers**: Use cloud services like AWS, Google Cloud, or Azure to run YOLOv5 and retrieve results via an API.
- **Local CPU Execution**: If no GPU is available, YOLOv5 can run on a CPU, though it will be slower.

CONNECTING TO YOLOV5 VIA THE INTERNET

Access YOLOv5 remotely without needing to install it locally:

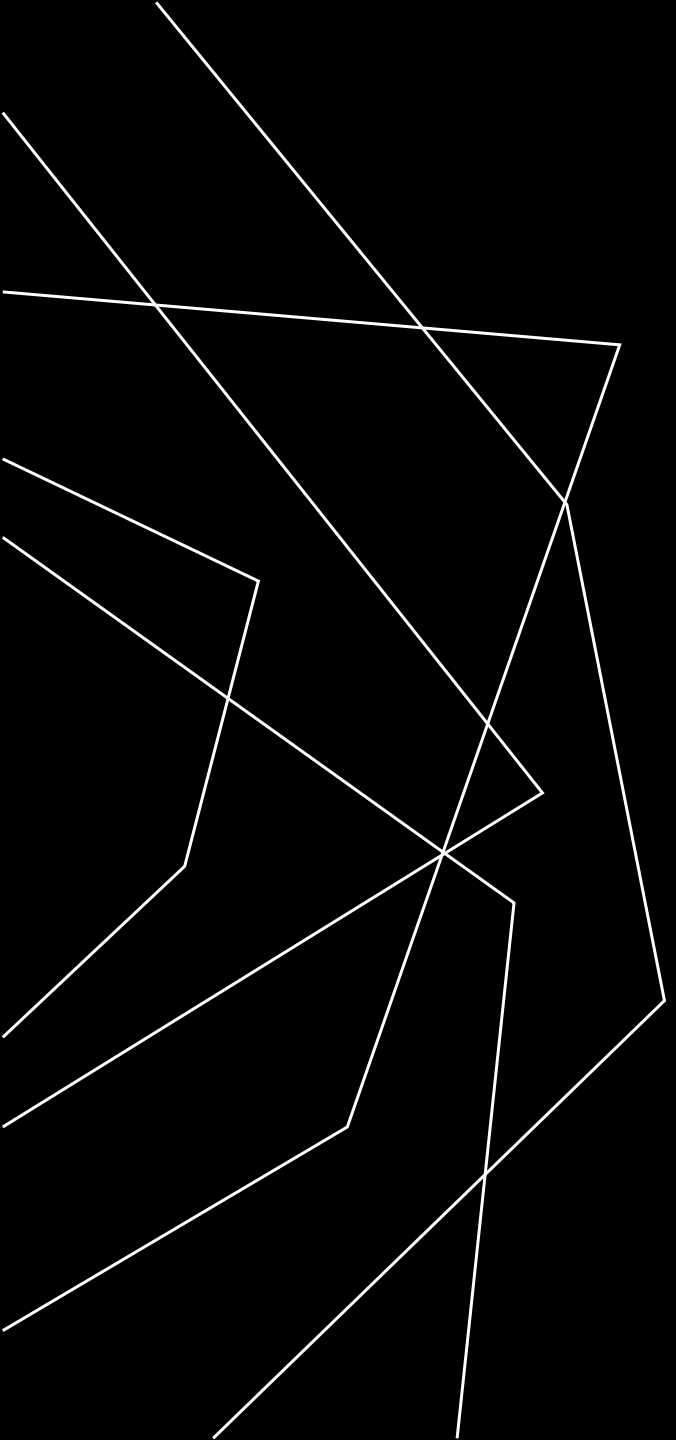
- **Use a Hosted YOLOv5 API:**
  - Roboflow provides an online YOLOv5 API where users can upload images and get detection results.
  - Users can also deploy their own API using **Flask** or **FastAPI** on a cloud server.
- **Pre-trained Model Weights:**
  - They can download and use pre-trained YOLOv5 models via GitHub or Hugging Face.
  - No training is required, just inference.

RUNNING OTHER COMPONENTS ON LAPTOPS

- **MQTT Communication**:
  - Can be tested locally using **Mosquitto** (lightweight and runs on any OS).
  - Cloud MQTT brokers (e.g., HiveMQ, Eclipse Mosquitto) allow users to connect remotely.
- **Simulation (Optional)**
  - **Gazebo/AirSim**: If needed, run simple drone simulations locally.
  - **Alternative**: work with 2D grid-based mapping using Python libraries like pygame.

RUNNING EVERYTHING ON LAPTOPS

- **Tools Used:**
  - **Object Detection**: Google Colab, online YOLOv5 API
  - **MQTT Messaging**: Mosquitto (local or cloud)
  - **Mapping & Visualization**: Python (matplotlib, pygame)
- **No Drones or Special Hardware Required!**
- **Scenario Examples:**
  - Simulating drone swarm object detection on laptops
  - Sharing detected object data



THANK YOU