Hybrid-Decentralized-Swarm-Intelligence-SI-Optimized-Intrusion-Detection-System-IDS

🛡️ Project Overview

This project implements a cutting-edge Hybrid Decentralized Intrusion Detection System (IDS) specifically designed for resource-constrained IoT environments. It leverages Particle Swarm Optimization (PSO) to efficiently tune the core classifier, resulting in high detection accuracy with ultra-low false alarms—critical for tactical or public IoT deployments.

🚀 Key Results

The optimization process achieved a significant performance balance by prioritizing the penalized fitness score (`Accuracy - 2 * FPR`).

Metric	PSO-Optimized RF	Vanilla RF Baseline	Target (Optimal for IoT)
Accuracy (ACC)	**95.1%**	92.3%	Maximize
False Positive Rate (FPR)	**3.8%**	5.2%	Minimize (Crucial)
Inference Time Reduction	**40%**	(N/A)	Maximize efficiency on edge nodes

🏗️ Two-Tier Architecture

The system operates on a decentralized model:

1. **Fog/Edge Node (Optimization Layer):**
   * Runs the computationally intensive **PSO algorithm** to find the optimal Random Forest hyperparameters and the most effective 12-feature subset.
   * Pushes the lightweight model configuration to the IoT Agents via **MQTT**.
2. **IoT Agent (The Swarm Layer):**
   * Hosts a pre-translated **C/C++ micro-model** (like a simple Decision Tree or Linear SVM).
   * Performs inference instantly using only the **12 optimized features**, resulting in $\mathbf{40\%}$ faster detection.

⚙️ Installation and Setup

A. Fog Node (Optimization)

1. **Prerequisites:** Python 3.8+, NSL-KDD dataset (`KDDTrain+.csv`, `KDDTest+.csv`).
2. **Dependencies:** Install required libraries:
   ```bash
   pip install pandas numpy scikit-learn pyswarms paho-mqtt joblib
   ```
3. **Run Optimization:**
   ```bash
   python optimization_engine.py
   ```
   This script will train the model and save the optimized feature mask and the model file (`optimized_model.pkl`).

B. Deployment Simulation (MQTT Testbed)

1. **Prerequisites:** An active MQTT Broker (e.g., Mosquitto).
2. **Run Simulation:**
   ```bash
   python mqtt_deployment_sim.py
   ```
   This runs the multi-threaded simulation, demonstrating model distribution and alert generation.

C. IoT Agent (Arduino Micro-Model)

1. **Model Translation:** Use a tool like `micromlgen` to convert the `optimized_model.pkl` into static C++ code.
2. **Deployment:** Upload the generated **`arduino_agent_micro_model.ino`** code to your target microcontroller (e.g., Arduino Uno or ESP32).

***

2. Python Code Files

A. `optimization_engine.py` (Fog Node)

(Contains the logic from the previous detailed response, including `load_data`, `penalized_fitness`, and `pso_optimize`.)

B. `mqtt_deployment_sim.py` (Deployment)

(Contains the logic from the previous detailed response, including `fog_publisher` and `agent_subscriber` threads.)

***

3. C++ Code File (Conceptual)

C. `arduino_agent_micro_model.ino`

(Contains the conceptual C++ code from the previous detailed response for the lightweight inference on the microcontroller.)