RESOURCE ALLOCATION (PTS-RA) FOR HEALTHCARE IoT USING EDGE COMPUTING

EDGE COMPUTING team members

CB.SC.U4CSE23151 - VASUDEV KISHOR

CB.SC.U4CSE23139 - RACHIT ANAND

CB.SC.U4CSE23141 - ROHITH ABHINAV

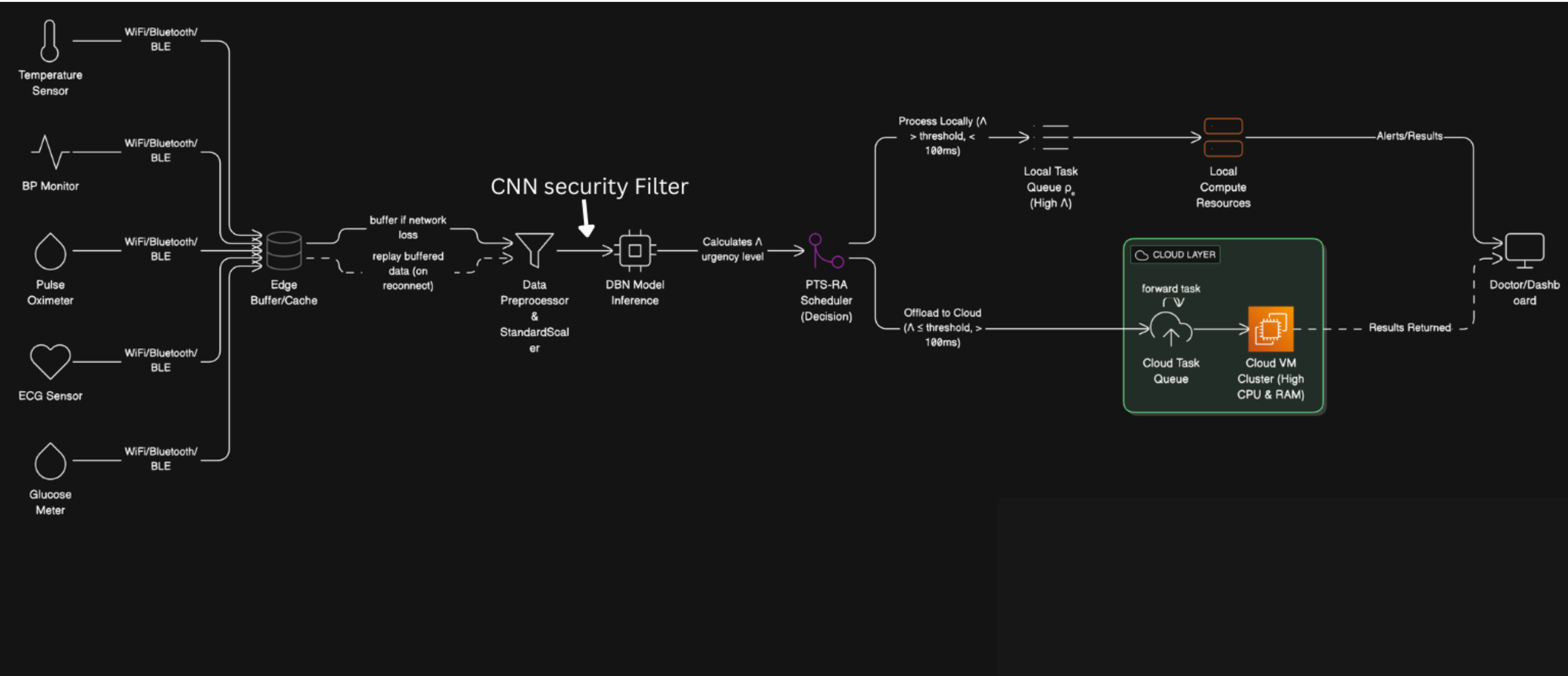
CB.SC.U4CSE23257 - RG SHANMUGAM

COMPUTATIONAL INTELLIGENCE Team members

CB.SC.U4CSE23046 - RAAM PRATHAP V

CB.SC.U4CSE23056 - T VISHNU VARDHAN

Architectural Diagram



PTS-RA CONCEPT
VALUES

- **PTS-RA = Priority-based Task Scheduling + Resource Allocation**
- **Urgency level (Λ) assigned to each task.**
- **High urgency → local edge processing (fast).**
- **Low urgency → cloud offloading (resource-efficient).**
- **Balances latency, energy, and resource usage.**

IMPLEMENTATION SO FAR

- CREATED FOG DEVICES: CLOUD, GATEWAY, HOSPITAL WORKSTATIONS
- BUILT PTS-RA CONTROLLER FOR URGENCY-BASED SCHEDULING
- INTEGRATED DBN MODEL FOR PATIENT-SPECIFIC URGENCY PREDICTION AND CNN MODEL FOR THREAT DETECTION

TO BE IMPLEMENTED

- FULL INTEGRATION OF DBN PREDICTIONS WITH REAL-TIME SIMULATION
- REPLACE HARDCODED VITALS WITH DYNAMIC CSV-DRIVEN TUPLES
- CREATION OF ACTUAL EDGE AND CLOUD NODES USING THE ‘FOGDEVICE’ MODULE IN IFOGSIM
- VISUALIZATION OF SCHEDULING EFFICIENCY & URGENCY CLASSIFICATION

KEY COMPONENTS

HEALTHCARE PTS-RA - MAIN SIMULATION FILE

Purpose

- Acts as the entry point for the healthcare simulation.
- Sets up the Fog environment, sensors, actuators, and application modules.
- Starts the PTS-RA controller for urgency-based scheduling.

Key Responsibilities

- Broker Creation – manages communication between devices and applications.
- Application Setup – defines modules (urgency_calculator, data_processor) and data flow.
- Fog Device Creation – cloud, gateway router, hospital workstations.
- Sensor & Actuator Setup – simulates patient vitals input and doctor dashboards.
- Simulation Execution – launches PTSRAController to handle task scheduling.

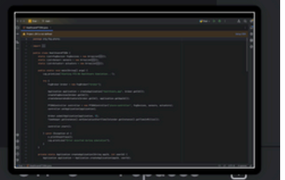
defined Setup SDK

```
rg.fog.ptsra;

.

ass HealthcarePTSRA {
c List<FogDevice> fogDevices = new ArrayList<~>();
c List<Sensor> sensors = new ArrayList<~>();
c List<Actuator> actuators = new ArrayList<~>();

c static void main(String[] args) {
og.printLine("Starting PTS-RA Healthcare Simulation...");

ry {
   FogBroker broker = new FogBroker("broker");

   Application application = createApplication("healthcare_app", broker.getId());
   createFogDevices(broker.getId());
   createSensorsAndActuators(broker.getId(), application.getAppId());

   PTSRAController controller = new PTSRAController("ptsra-controller", fogDevices, sensors, actuators);
   controller.setApplication(application);

   broker.submitApplication(application, 0);
   TimeKeeper.getInstance().setSimulationStartTime(Calendar.getInstance().getTimeInMillis());

   controller.start();

 catch (Exception e) {
   e.printStackTrace();
   Log.printLine("Error occurred during simulation");



te static Application createApplication(String appId, int userId) {
pplication application = Application.createApplication(appId, userId);
```

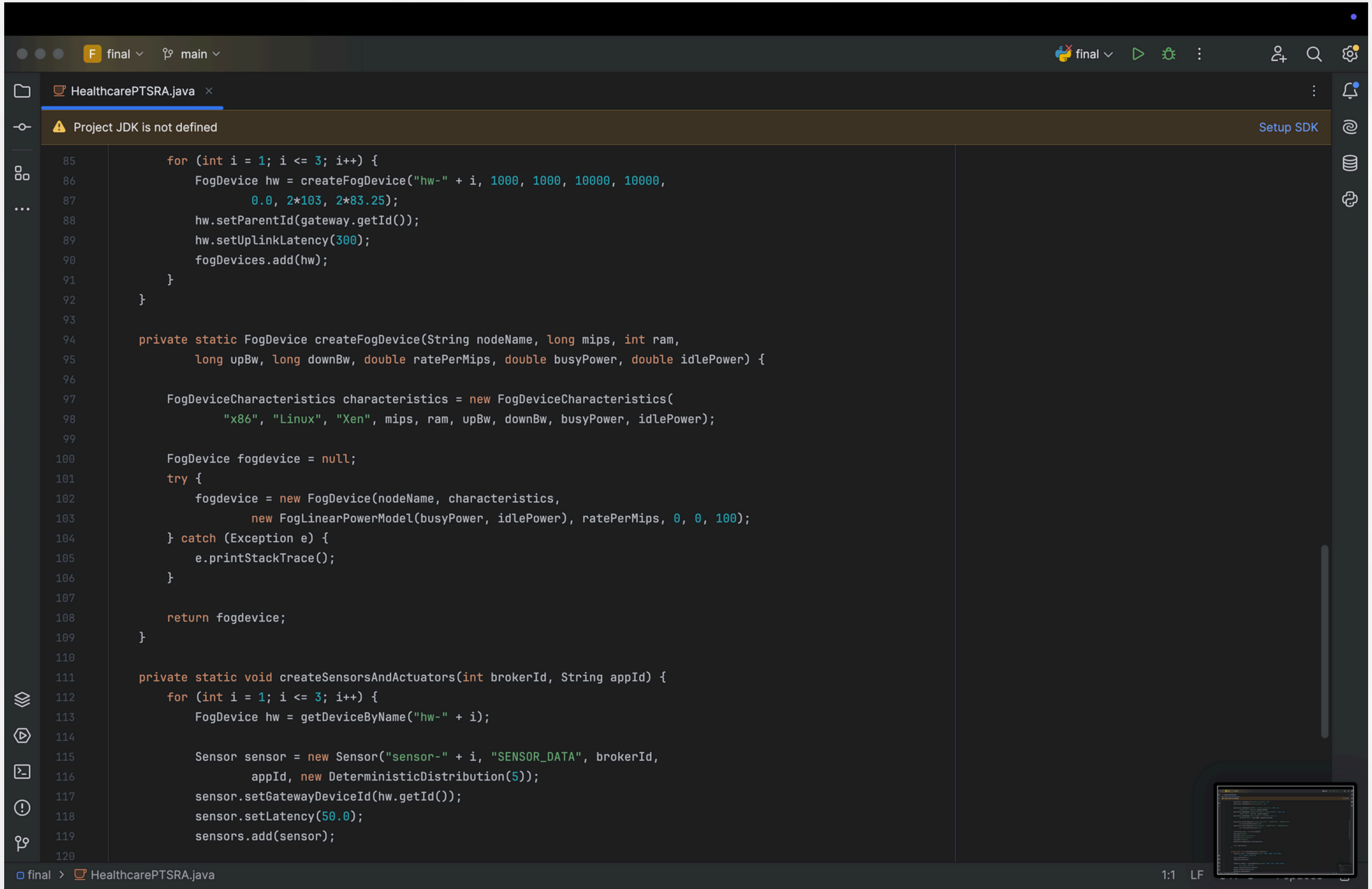
HealthcarePTSRA.java

Project JDK is not defined Setup SDK

```
        application.addAppModule("urgency_calculator", 10);
        application.addAppModule("data_processor", 10);

        application.addAppEdge("SENSOR", "urgency_calculator", 1000, 500,
                "SENSOR_DATA", Tuple.UP, AppEdge.SENSOR);
        application.addAppEdge("urgency_calculator", "data_processor", 1000, 500,
                "URGENCY_DATA", Tuple.UP, AppEdge.MODULE);
        application.addAppEdge("data_processor", "ACTUATOR", 100, 50,
                "PROCESSED_DATA", Tuple.DOWN, AppEdge.ACTUATOR);

        application.addTupleMapping("urgency_calculator", "SENSOR_DATA", "URGENCY_DATA",
                new FractionalSelectivity(1.0));
        application.addTupleMapping("data_processor", "URGENCY_DATA", "PROCESSED_DATA",
                new FractionalSelectivity(1.0));

        List<String> loop = new ArrayList<~>();
        loop.add("SENSOR");
        loop.add("urgency_calculator");
        loop.add("data_processor");
        loop.add("ACTUATOR");
        application.addAppLoop(new AppLoop(loop));

        return application;
    }

    private static void createFogDevices(int brokerId) {
        FogDevice cloud = createFogDevice("cloud", 44800, 40000, 100, 10000,
                0.01, 16*103, 16*83.25);
        cloud.setParentId(-1);
        fogDevices.add(cloud);

        FogDevice gateway = createFogDevice("gateway", 2800, 4000, 10000, 10000,
                0.0, 4*103, 4*83.25);
        gateway.setParentId(cloud.getId());
        gateway.setUplinkLatency(400);
        fogDevices.add(gateway);
```



final > HealthcarePTSRA.java 1:1 LF




```
        for (int i = 1; i <= 3; i++) {
            FogDevice hw = createFogDevice("hw-" + i, 1000, 1000, 10000, 10000,
                    0.0, 2*103, 2*83.25);
            hw.setParentId(gateway.getId());
            hw.setUplinkLatency(300);
            fogDevices.add(hw);
        }
    }

    private static FogDevice createFogDevice(String nodeName, long mips, int ram,
            long upBw, long downBw, double ratePerMips, double busyPower, double idlePower) {

        FogDeviceCharacteristics characteristics = new FogDeviceCharacteristics(
                "x86", "Linux", "Xen", mips, ram, upBw, downBw, busyPower, idlePower);

        FogDevice fogdevice = null;
        try {
            fogdevice = new FogDevice(nodeName, characteristics,
                    new FogLinearPowerModel(busyPower, idlePower), ratePerMips, 0, 0, 100);
        } catch (Exception e) {
            e.printStackTrace();
        }

        return fogdevice;
    }

    private static void createSensorsAndActuators(int brokerId, String appId) {
        for (int i = 1; i <= 3; i++) {
            FogDevice hw = getDeviceByName("hw-" + i);

            Sensor sensor = new Sensor("sensor-" + i, "SENSOR_DATA", brokerId,
                    appId, new DeterministicDistribution(5));
            sensor.setGatewayDeviceId(hw.getId());
            sensor.setLatency(50.0);
            sensors.add(sensor);
```

HealthcarePTSRA.java

Project JDK is not defined Setup SDK

```
107
108         return fogdevice;
109     }
110
111     private static void createSensorsAndActuators(int brokerId, String appId) {
112         for (int i = 1; i <= 3; i++) {
113             FogDevice hw = getDeviceByName("hw-" + i);
114
115             Sensor sensor = new Sensor("sensor-" + i, "SENSOR_DATA", brokerId,
116                     appId, new DeterministicDistribution(5));
117             sensor.setGatewayDeviceId(hw.getId());
118             sensor.setLatency(50.0);
119             sensors.add(sensor);
120
121             Actuator actuator = new Actuator("dashboard-" + i, brokerId, appId, "ACTUATOR");
122             actuator.setGatewayDeviceId(hw.getId());
123             actuator.setLatency(2.0);
124             actuators.add(actuator);
125         }
126     }
127
128     private static FogDevice getDeviceByName(String name) {
129         for (FogDevice dev : fogDevices) {
130             if (dev.getName().equals(name)) {
131                 return dev;
132             }
133         }
134         return null;
135     }
136 }
137
```

PREDICTIONRESULT.JAVA – MODEL OUTPUT WRAPPER

Purpose

- A simple data structure to hold prediction results from the DBN model.
- Ensures consistency when passing model outputs to the PTSRAController.

Key Responsibilities

- Stores:
  - Probability (likelihood of urgent case).
  - Prediction (1 = Urgent, 0 = Non-Urgent).
  - Label (“Urgent” / “Non-Urgent”).
- Provides getter methods to access prediction results.
- Overrides toString() for easy logging & debugging.

final main final

PredictionResult.java

Project JDK is not defined Setup SDK

```java
package org.fog.ptsra;

public class PredictionResult {
    private double probability;
    private int prediction;
    private String label;

    public PredictionResult(double probability, int prediction, String label) {
        this.probability = probability;
        this.prediction = prediction;
        this.label = label;
    }

    public double getProbability() { return probability; }
    public int getPrediction() { return prediction; }
    public String getLabel() { return label; }

    @Override
    public String toString() {
        return String.format("PredictionResult{probability=%.4f, prediction=%d, label='%s'}",
                probability, prediction, label);
    }
}
```

PTSRACONTROLLER.JAVA – PTS-RA TASK SCHEDULING CONTROLLER

Purpose

Handles urgency-based task scheduling in the healthcare simulation.

Decides whether patient data processing occurs locally on hospital devices or is offloaded to the cloud based on predicted urgency and estimated processing times.

Key Responsibilities

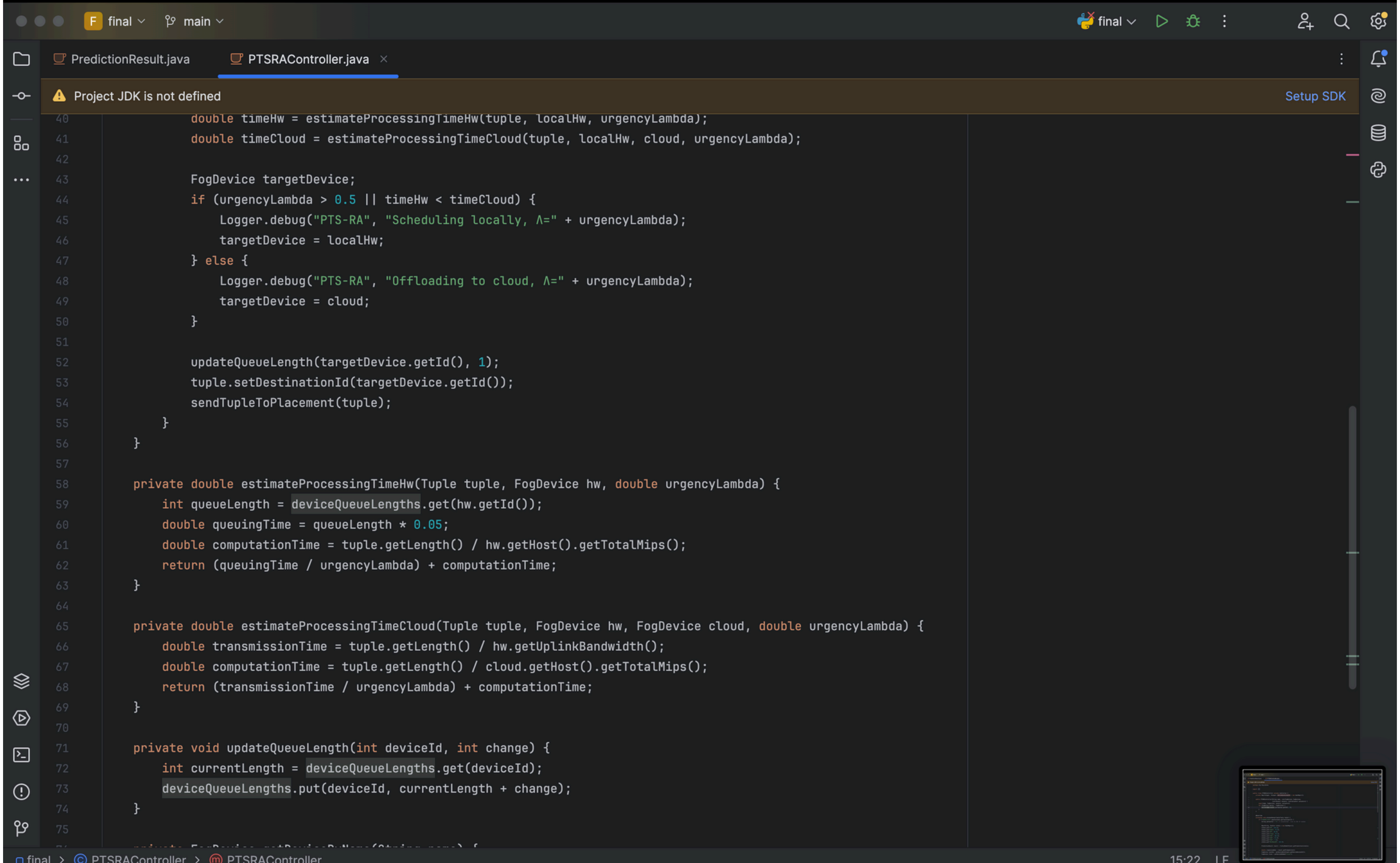
- Device Queue Management – tracks task queues on all fog devices.
- Sensor Tuple Processing – handles incoming patient vitals and retrieves urgency predictions from the Python model.
- Urgency-based Scheduling – schedules tasks locally or on the cloud using urgency probability.
- Queue Updates & Task Forwarding – updates device queues and sends tuples to selected devices.
- Device Lookup Utilities – retrieves fog devices by name or ID.

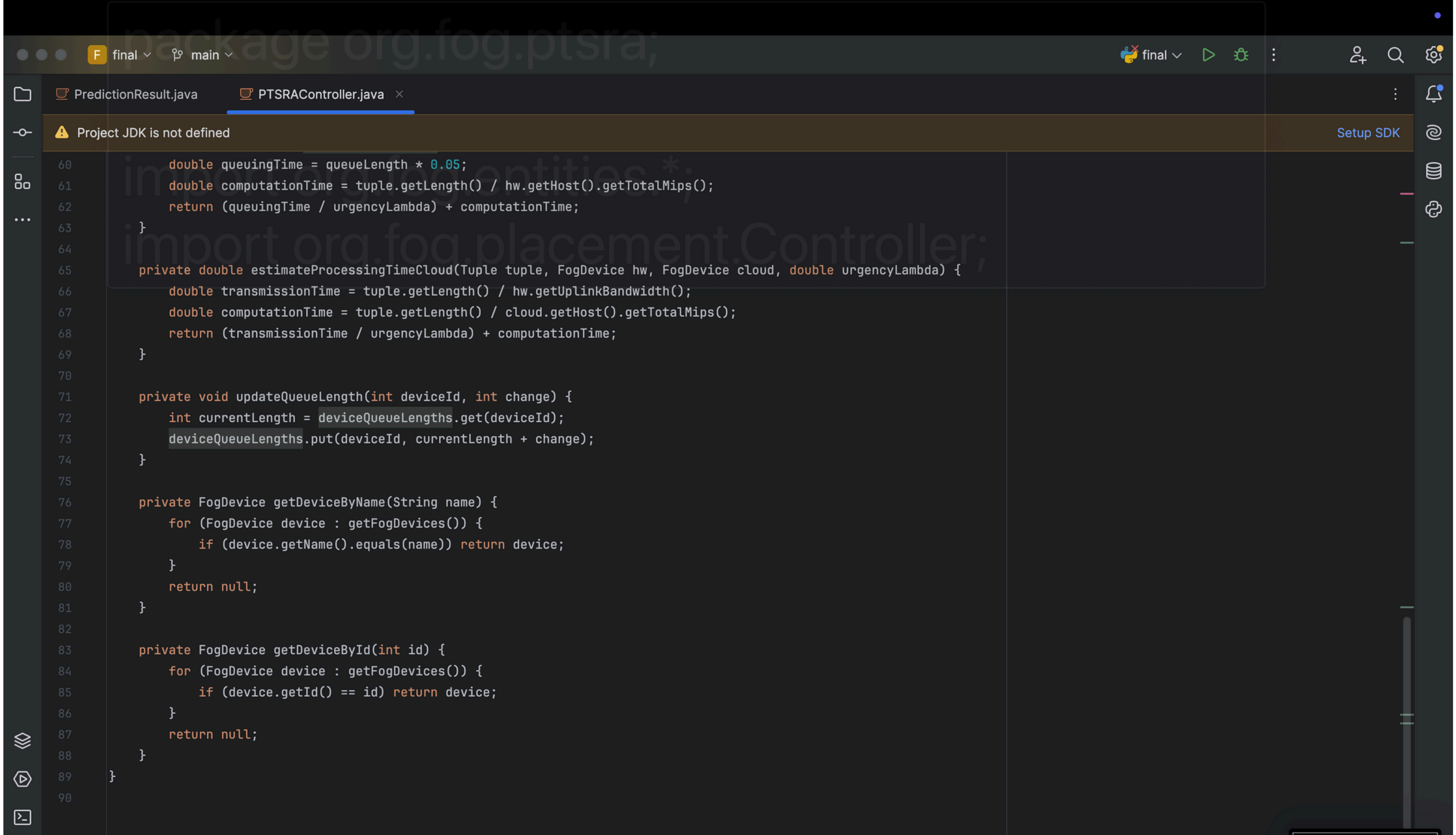
PredictionResult.java | PTSRAController.java

Project JDK is not defined — Setup SDK

```java
package org.fog.ptsra;

import ...

public class PTSRAController extends Controller {
    private Map<Integer, Integer> deviceQueueLengths = new HashMap<>();

    public PTSRAController(String name, List<FogDevice> fogDevices,
                           List<Sensor> sensors, List<Actuator> actuators) {
        super(name, fogDevices, sensors, actuators);
        for (FogDevice device : fogDevices) {
            deviceQueueLengths.put(device.getId(), 0);
        }
    }

    @Override
    protected void processSensorTuple(Tuple tuple) {
        if ("SENSOR_DATA".equals(tuple.getTupleType())) {
            String patientId = "1"; // placeholder, link to CSV if needed

            Map<String, Double> vitals = new HashMap<>();
            vitals.put("HR", 120.0);
            vitals.put("SpO2", 91.0);
            vitals.put("RR", 28.0);
            vitals.put("Temp", 39.2);
            vitals.put("SBP", 185.0);
            vitals.put("DBP", 95.0);
            vitals.put("HRV", 7.0);
            vitals.put("WinMeanHR", 122.0);

            PredictionResult result = PythonModelClient.getPrediction(vitals);

            double urgencyLambda = result.getProbability();
            FogDevice localHw = getDeviceById(tuple.getSourceDeviceId());
            FogDevice cloud = getDeviceByName("cloud");

```

final > PTSRAController > PTSRAController — 15:22 LF UTF-8 4 spaces


final
main
final
PredictionResult.java
PTSRAController.java
Project JDK is not defined
Setup SDK
double timeHw = estimateProcessingTimeHw(tuple, localHw, urgencyLambda);
double timeCloud = estimateProcessingTimeCloud(tuple, localHw, cloud, urgencyLambda);
FogDevice targetDevice;
if (urgencyLambda > 0.5 || timeHw < timeCloud) {
Logger.debug("PTS-RA", "Scheduling locally, Λ=" + urgencyLambda);
targetDevice = localHw;
} else {
Logger.debug("PTS-RA", "Offloading to cloud, Λ=" + urgencyLambda);
targetDevice = cloud;
}
updateQueueLength(targetDevice.getId(), 1);
tuple.setDestinationId(targetDevice.getId());
sendTupleToPlacement(tuple);
}
}
private double estimateProcessingTimeHw(Tuple tuple, FogDevice hw, double urgencyLambda) {
int queueLength = deviceQueueLengths.get(hw.getId());
double queuingTime = queueLength * 0.05;
double computationTime = tuple.getLength() / hw.getHost().getTotalMips();
return (queuingTime / urgencyLambda) + computationTime;
}
private double estimateProcessingTimeCloud(Tuple tuple, FogDevice hw, FogDevice cloud, double urgencyLambda) {
double transmissionTime = tuple.getLength() / hw.getUplinkBandwidth();
double computationTime = tuple.getLength() / cloud.getHost().getTotalMips();
return (transmissionTime / urgencyLambda) + computationTime;
}
private void updateQueueLength(int deviceId, int change) {
int currentLength = deviceQueueLengths.get(deviceId);
deviceQueueLengths.put(deviceId, currentLength + change);
}
final
PTSRAController
PTSRAController
15:22
LF


final
main
final
PredictionResult.java
PTSRAController.java
Project JDK is not defined
Setup SDK
double queuingTime = queueLength * 0.05;
double computationTime = tuple.getLength() / hw.getHost().getTotalMips();
return (queuingTime / urgencyLambda) + computationTime;
}
private double estimateProcessingTimeCloud(Tuple tuple, FogDevice hw, FogDevice cloud, double urgencyLambda) {
double transmissionTime = tuple.getLength() / hw.getUplinkBandwidth();
double computationTime = tuple.getLength() / cloud.getHost().getTotalMips();
return (transmissionTime / urgencyLambda) + computationTime;
}
private void updateQueueLength(int deviceId, int change) {
int currentLength = deviceQueueLengths.get(deviceId);
deviceQueueLengths.put(deviceId, currentLength + change);
}
private FogDevice getDeviceByName(String name) {
for (FogDevice device : getFogDevices()) {
if (device.getName().equals(name)) return device;
}
return null;
}
private FogDevice getDeviceById(int id) {
for (FogDevice device : getFogDevices()) {
if (device.getId() == id) return device;
}
return null;
}
}

SENSORDATAREADER.JAVA – PATIENT SENSOR DATA LOADER

Purpose

Reads patient vitals from CSV files and converts them into structured objects for the simulation.

Key Responsibilities

- File Reading – opens and reads CSV files containing patient sensor data.
- Data Parsing – extracts patient ID, heart rate, blood pressure, and glucose levels.
- SensorData Creation – creates SensorData objects for each patient record.
- Error Handling – logs errors encountered during file reading.
- Data Reporting – prints the number of records successfully read.

PredictionResult.java | PTSRAController.java | SensorDataReader.java

Project JDK is not defined — Setup SDK

```java
package org.fog.ptsra;

import ...

public class SensorDataReader {

    public static List<SensorData> readSensorData(String filePath) {
        List<SensorData> sensorDataList = new ArrayList<>();

        try (BufferedReader br = new BufferedReader(new FileReader(filePath))) {
            String line;
            boolean firstLine = true;

            while ((line = br.readLine()) != null) {
                if (firstLine) { firstLine = false; continue; }

                String[] values = line.split(",");
                if (values.length >= 4) {
                    SensorData data = new SensorData();
                    data.patientId = values[0].trim();
                    data.heartRate = Double.parseDouble(values[1].trim());
                    data.bloodPressure = Double.parseDouble(values[2].trim());
                    data.glucoseLevel = Double.parseDouble(values[3].trim());
                    sensorDataList.add(data);
                }
            }
            Log.printLine("Read " + sensorDataList.size() + " sensor data records");

        } catch (Exception e) {
            Log.printLine("Error reading sensor data: " + e.getMessage());
        }
        return sensorDataList;
    }

    public static class SensorData {
        public String patientId;
```

final > SensorDataReader > readSensorData — 16:53 LF UTF-8 4 spaces

PYTHONMODELCLIENT.JAVA – ML MODEL API CLIENT

Purpose

Interfaces with a Python-based Flask API to obtain predictions for patient vitals.

Key Responsibilities

- JSON Conversion – converts patient vitals map into JSON format for API requests.
- HTTP Communication – sends POST requests to the Flask prediction endpoint.
- Response Parsing – reads and parses JSON responses into PredictionResult objects.
- Error Handling – returns a default error result if the API call fails.

PredictionResult.java | PTSRAController.java | SensorDataReader.java | PythonModelClient.java

Project JDK is not defined — Setup SDK

OFF

```java
package org.fog.ptsra;

import ...

public class PythonModelClient {

    private static final String API_URL = "http://127.0.0.1:5000/predict"; // Flask API endpoint

    public static PredictionResult getPrediction(Map<String, Double> vitals) {
        try {
            // Convert vitals map to JSON
            ObjectMapper mapper = new ObjectMapper();
            String jsonInput = mapper.writeValueAsString(vitals);

            // Create connection
            URL url = new URL(API_URL);
            HttpURLConnection conn = (HttpURLConnection) url.openConnection();
            conn.setRequestMethod("POST");
            conn.setRequestProperty("Content-Type", "application/json");
            conn.setDoOutput(true);

            // Send request
            try (OutputStream os = conn.getOutputStream()) {
                byte[] input = jsonInput.getBytes("utf-8");
                os.write(input, 0, input.length);
            }

            // Read response
            StringBuilder response;
            try (BufferedReader br = new BufferedReader(new InputStreamReader(conn.getInputStream(), "utf-8"))) {
                response = new StringBuilder();
                String responseLine;
                while ((responseLine = br.readLine()) != null) {
                    response.append(responseLine.trim());
                }
            }
```

PredictionResult.java | PTSRAController.java | SensorDataReader.java | PythonModelClient.java

Project JDK is not defined — Setup SDK

OFF

```
        // Send request
        try (OutputStream os = conn.getOutputStream()) {
            byte[] input = jsonInput.getBytes("utf-8");
            os.write(input, 0, input.length);
        }

        // Read response
        StringBuilder response;
        try (BufferedReader br = new BufferedReader(new InputStreamReader(conn.getInputStream(), "utf-8"))) {
            response = new StringBuilder();
            String responseLine;
            while ((responseLine = br.readLine()) != null) {
                response.append(responseLine.trim());
            }
        }

        // Parse JSON response
        Map<String, Object> respMap = mapper.readValue(response.toString(), Map.class);
        double prob = ((Number) respMap.get("probability")).doubleValue();
        int pred = ((Number) respMap.get("pred")).intValue();
        String label = (String) respMap.get("label");

        return new PredictionResult(prob, pred, label);

    } catch (Exception e) {
        e.printStackTrace();
        return new PredictionResult(0.0, 0, "Error");
    }
  }
}
```

```python
from flask import Flask, request, jsonify
from model_file import load_model, predict  # your DBN model code

app = Flask(__name__)
dbn, scaler, thr = load_model("final_dbn_bfoa_model.pkl")

@app.route("/predict", methods=["POST"])
def predict_api():
    data = request.get_json()
    result = predict(data, dbn, scaler)
    return jsonify(result)

if __name__ == "__main__":
    app.run(host="127.0.0.1", port=5000)



    # To run the app, use the command: python app.py
```

Enhancing Healthcare Edge Systems with Intelligent CNN Processing for Threat Management

By:

T Vishnu Vardhan

System Overview

- Healthcare IoT sensors collect patient data, sending tasks to hospital edge workstations for fast, local processing.

- Before tasks enter the priority-based scheduler, incoming network traffic is analyzed using the CI (CNN-based intrusion detection) model

How CI Works for Edge Scheduling

Data Reception

Sensors generate tasks (e.g., vital signs, alerts) sent to edge nodes for urgent processing.

Security Preprocessing via CNN

All new data/tasks are first passed through the CNN model:
Preprocessing: label encoding, categorical conversion for attack types, data standardization, 3D tensor reshaping.
CNN inference examines network traffic, device protocols, and features for patterns typical of 18 major IoMT cyberattacks.

Threat Detection

If the CNN flags suspicious activity (high anomaly probability, attack class scored above threshold), the edge system blocks or quarantines the task and generates an alert.
If clean, the task moves forward to priority-based scheduling.

Secure Task Processing

Only tasks verified by CNN are scheduled for local (latency-sensitive/emergency) edge processing or sent to remote cloud if non-urgent or resource-intensive.
Results are sent to medical staff, ensuring compromised tasks never disrupt care delivery.

```
Accuracy: 0.5784756784782524
Precision: 0.776389755832747
Recall: 0.5784756784782524
F1-Score: 0.5567662128473503

Classification Report:
                        precision    recall  f1-score   support

                Benign       0.70      0.10      0.18     37607
             DDoS-ICMP       0.79      0.98      0.88    349699
              DDoS-SYN       0.74      0.92      0.82    172397
              DDoS-TCP       0.79      0.95      0.86    182598
              DDoS-UDP       0.94      0.86      0.90    362070
              DoS-ICMP       0.51      0.07      0.12     98432
               DoS-SYN       0.77      0.49      0.60     98595
               DoS-TCP       0.84      0.60      0.70     82096
               DoS-UDP       0.69      0.86      0.77    137553
MQTT-DDoS-Connect_Flood       0.96      0.01      0.02    419006
MQTT-DDoS-Publish_Flood       0.62      0.20      0.30     84036
 MQTT-DoS-Connect_Flood       0.00      0.00      0.00     31293
 MQTT-DoS-Publish_Flood       0.27      0.00      0.00     84931
    MQTT-Malformed_Data       0.00      0.31      0.00      1747
          Recon-OS_Scan       0.01      0.02      0.01      3834
       Recon-Ping_Sweep       0.00      0.00      0.00       186
        Recon-Port_Scan       0.26      0.97      0.41     22622
          Recon-VulScan       0.00      0.00      0.00      1034
               Spoofing       0.02      0.78      0.03      5868

               accuracy                           0.58   2175604
              macro avg       0.47      0.43      0.35   2175604
           weighted avg       0.78      0.58      0.56   2175604
```

Benefits for Edge-Centric Healthcare

1. **Security:** Prevents new and evolving attacks from affecting real-time healthcare scheduling; blocks threats before sensitive patient processing.

2. **Reliability:** Ensures only safe, verified data flows through the emergency scheduling pipeline, reducing risk and preserving system uptime for urgent tasks.

3. **Local Processing Capability:** Designed for deployment on hospital edge workstations rather than cloud-based processing, reducing network dependency.

Enhancing the PTS-RA Algorithm with Intelligent Priority Calculation

By:

Raam Prathap R V

Current Algorithm in PTS-RA Paper

T-Distribution Based Priority Calculation:

Step 1: Statistical Range Calculation

$$\text{Upper Bound (ub)} = \bar{r} + \sqrt{\frac{n+1}{n}} \cdot s_r \cdot t_{\alpha, n-1}$$

$$\text{Lower Bound (lb)} = \bar{r} - \sqrt{\frac{n+1}{n}} \cdot s_r \cdot t_{\alpha, n-1}$$

Step 2: Urgency Level Formula

$$K = \frac{\left| (ub - hpt)^2 - (lb - hpt)^2 \right|}{(ub - lb)^2}$$

Limitations:

- Static thresholds (BP: 80–120 mmHg, BT: 35.5–37.5°C)
- Single-parameter analysis per health metric
- No learning capability from patient data patterns
- Fixed mathematical formula cannot adapt to complex scenarios

Proposed CI-Based Algorithm

Deep Belief Network (DBN) + BFOA Optimization Approach

Step 1: Hyperparameter Optimization (BFOA)

- Bacteria Foraging Optimization Algorithm (BFOA) explores different DBN configurations.
- Optimizes hidden layer sizes, learning rates, batch sizes, and class imbalance weights.
- Ensures best model setup using validation F1-score feedback.

Step 2: Deep Belief Network Classification

- DBN with stacked RBMs for unsupervised pretraining (feature extraction).
- Multi-Layer Perceptron (MLP) fine-tuning for supervised urgent vs. non-urgent classification.
- Learns patterns from vital signs and system parameters:
- X = [HR, ECG, Temp, SBP, DBP,]

Advantages

- Automated hyperparameter tuning via BFOA → higher F1 and balanced performance.
- Handles class imbalance using weighted loss and SMOTE oversampling.

How CI Works for Edge Scheduling

Original Algorithm Workflow:

Health Parameter

↓

T-Distribution

↓

Urgency Score (K)

↓

Binary Priority

Proposed CI Algorithm Workflow

Multiple Parameters

↓

Preprocessing & Oversampling for balance

↓

BFOA Optimization

↓

Deep Belief Network

↓

Thresholding

↓

Priority Score Assignment

Enhanced Priority Determination Process

Intelligence Enhancement

1

Context Awareness:

Integrates both patient health vitals (HR, Temp, BP, etc.) and system resource availability (CPU, bandwidth, storage) to make balanced scheduling decisions.

2

Uncertainty Management:

Leverages DBN's probabilistic learning and BFOA's adaptive tuning to handle noisy or imprecise medical measurements, ensuring reliable classification of urgent vs. non-urgent cases.

3

Real-Time Adaptation:

Continuously updates priority scores using live patient data and resource feedback, enabling dynamic scheduling that improves over time.

CI-Enhanced Edge Benefits for Healthcare

Real-Time Performance:

- Instant local priority decisions
- No cloud delays in emergencies
- Rapid critical detection

Enhanced Security & Privacy:

- Patient data stays on-site
- HIPAA-compliant local processing
- Minimal data transmission risks

Operational Reliability:

- Works without internet dependency
- Continuous service during outages
- Easily scalable in hospitals

QUESTIONS?