

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

EDGE COMPUTING team members

CB.SC.U4CSE23151 - VASUDEV KISHOR

CB.SC.U4CSE23139 - RACHIT ANAND

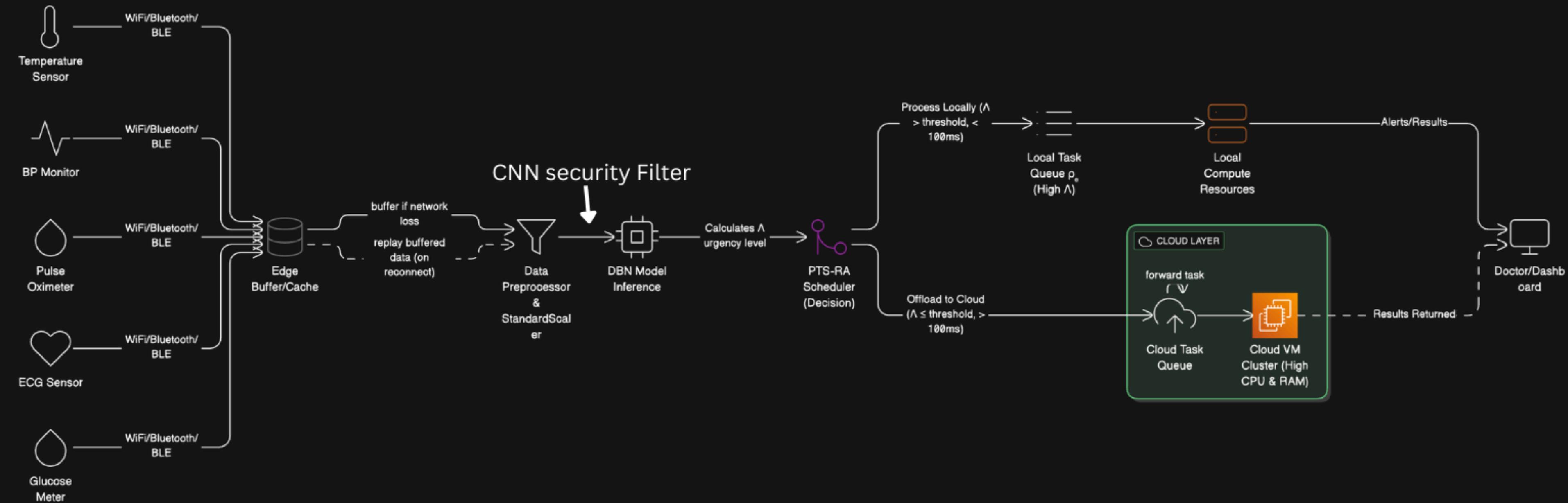
CB.SC.U4CSE23141 - ROHITH ABHINAV

COMPUTATIONAL INTELLIGENCE Team members

CB.SC.U4CSE23046 - Raam Prathap R V

CB.SC.U4CSE23056 - T Vishnu Vardhan

Architectural Diagram



PTS-RA CONCEPT **VALUES**

- **PTS-RA = Priority-based Task Scheduling + Resource Allocation**
- **Urgency level (\wedge) assigned to each task.**
- **High urgency \rightarrow local edge processing (fast).**
- **Low urgency \rightarrow 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

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.

The screenshot shows a Java IDE interface with a dark theme. The top bar has tabs for 'final' and 'main'. The main area displays a Java file named 'HealthcarePTSRA.java'. A prominent yellow warning bar at the top states 'Project JDK is not defined' with a 'Setup SDK' link. The code itself is a class definition for 'HealthcarePTSRA' with several methods and static blocks. The left margin shows line numbers from 85 to 120. On the far left, there's a vertical toolbar with icons for file operations like copy, paste, and search, as well as a help icon.

```
85     for (int i = 1; i <= 3; i++) {
86         FogDevice hw = createFogDevice("hw-" + i, 1000, 1000, 10000, 10000,
87                                         0.0, 2*103, 2*83.25);
88         hw.setParentId(gateway.getId());
89         hw.setUplinkLatency(300);
90         fogDevices.add(hw);
91     }
92 }
93
94 private static FogDevice createFogDevice(String nodeName, long mips, int ram,
95                                         long upBw, long downBw, double ratePerMips, double busyPower, double idlePower) {
96
97     FogDeviceCharacteristics characteristics = new FogDeviceCharacteristics(
98         "x86", "Linux", "Xen", mips, ram, upBw, downBw, busyPower, idlePower);
99
100    FogDevice fogdevice = null;
101    try {
102        fogdevice = new FogDevice(nodeName, characteristics,
103                               new FogLinearPowerModel(busyPower, idlePower), ratePerMips, 0, 0, 100);
104    } catch (Exception e) {
105        e.printStackTrace();
106    }
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 }
```

HealthcarePTSRA.java

Project JDK is not defined

Setup SDK

```
1 package org.fog.ptsra;
2
3 > import ...
...
15
16 public class HealthcarePTSRA {
17     static List<FogDevice> fogDevices = new ArrayList<~>();
18     static List<Sensor> sensors = new ArrayList<~>();
19     static List<Actuator> actuators = new ArrayList<~>();
20
21     public static void main(String[] args) {
22         Log.printLine("Starting PTS-RA Healthcare Simulation...");
23
24     try {
25         FogBroker broker = new FogBroker("broker");
26
27         Application application = createApplication("healthcare_app", broker.getId());
28         createFogDevices(broker.getId());
29         createSensorsAndActuators(broker.getId(), application.getAppId());
30
31         PTSRAController controller = new PTSRAController("ptsra-controller", fogDevices, sensors, actuators);
32         controller.setApplication(application);
33
34         broker.submitApplication(application, 0);
35         TimeKeeper.getInstance().setSimulationStartTime(Calendar.getInstance().getTimeInMillis());
36
37         controller.start();
38
39     } catch (Exception e) {
40         e.printStackTrace();
41         Log.printLine("Error occurred during simulation");
42     }
43 }
44
45     private static Application createApplication(String appId, int userId) {
46         Application application = Application.createApplication(appId, userId);
47 }
```



HealthcarePTSRA.java ×



Project JDK is not defined

Setup SDK

```
48     application.addAppModule("urgency_calculator", 10);
49     application.addAppModule("data_processor", 10);
...
51     application.addAppEdge("SENSOR", "urgency_calculator", 1000, 500,
52         "SENSOR_DATA", Tuple.UP, AppEdge.SENSOR);
53     application.addAppEdge("urgency_calculator", "data_processor", 1000, 500,
54         "URGENCY_DATA", Tuple.UP, AppEdge.MODULE);
55     application.addAppEdge("data_processor", "ACTUATOR", 100, 50,
56         "PROCESSED_DATA", Tuple.DOWN, AppEdge.ACTUATOR);

57
58     application.addTupleMapping("urgency_calculator", "SENSOR_DATA", "URGENCY_DATA",
59         new FractionalSelectivity(1.0));
60     application.addTupleMapping("data_processor", "URGENCY_DATA", "PROCESSED_DATA",
61         new FractionalSelectivity(1.0));

62     List<String> loop = new ArrayList<~>();
63     loop.add("SENSOR");
64     loop.add("urgency_calculator");
65     loop.add("data_processor");
66     loop.add("ACTUATOR");
67     application.addAppLoop(new AppLoop(loop));

68
69     return application;
70 }

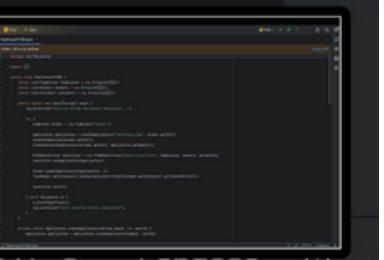
71 }

72
73 private static void createFogDevices(int brokerId) {
74     FogDevice cloud = createFogDevice("cloud", 44800, 40000, 100, 10000,
75         0.01, 16*103, 16*83.25);
76     cloud.setParentId(-1);
77     fogDevices.add(cloud);

78
79     FogDevice gateway = createFogDevice("gateway", 2800, 4000, 10000, 10000,
80         0.0, 4*103, 4*83.25);
81     gateway.setParentId(cloud.getId());
82     gateway.setUplinkLatency(400);
83     fogDevices.add(gateway);
```

final > HealthcarePTSRA.java

1:1 LF





HealthcarePTSRA.java



Project JDK is not defined

Setup SDK

```
107         return fogdevice;
108     }
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.

The screenshot shows a Java development environment with the following details:

- Title Bar:** The title bar displays "final" and "main" with dropdown arrows.
- Project Status:** A message "Project JDK is not defined" is shown in the top right corner.
- Code Editor:** The main area contains the code for `PredictionResult.java`. The code defines a class with private fields for probability, prediction, and label, and public methods to get each. It also overrides the `toString()` method to format the object as a string.

```
1 package org.fog ptsra;
2
3 public class PredictionResult {
4     private double probability;
5     private int prediction;
6     private String label;
7
8     public PredictionResult(double probability, int prediction, String label) {
9         this.probability = probability;
10        this.prediction = prediction;
11        this.label = label;
12    }
13
14    public double getProbability() { return probability; }
15    public int getPrediction() { return prediction; }
16    public String getLabel() { return label; }
17
18    @Override
19    public String toString() {
20        return String.format("PredictionResult{probability=%.4f, prediction=%d, label='%s'}",
21                            probability, prediction, label);
22    }
23 }
24
```

- SIDE BAR:** On the left side, there are icons for file operations like Open, Save, and Close, along with a search and settings icon.

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

```
1 package org.fog.ptsla;
2
3 > import ...
...
7
8 public class PTSRAController extends Controller {
9     private Map<Integer, Integer> deviceQueueLengths = new HashMap<>();
10
11     public PTSRAController(String name, List<FogDevice> fogDevices,
12                           List<Sensor> sensors, List<Actuator> actuators) {
13         super(name, fogDevices, sensors, actuators);
14         for (FogDevice device : fogDevices) {
15             deviceQueueLengths.put(device.getId(), 0);
16         }
17     }
18
19     @Override
20     protected void processSensorTuple(Tuple tuple) {
21         if ("SENSOR_DATA".equals(tuple.getTupleType())) {
22             String patientId = "1"; // placeholder, link to CSV if needed
23
24             Map<String, Double> vitals = new HashMap<>();
25             vitals.put("HR", 120.0);
26             vitals.put("SpO2", 91.0);
27             vitals.put("RR", 28.0);
28             vitals.put("Temp", 39.2);
29             vitals.put("SBP", 185.0);
30             vitals.put("DBP", 95.0);
31             vitals.put("HRV", 7.0);
32             vitals.put("WinMeanHR", 122.0);
33
34             PredictionResult result = PythonModelClient.getPrediction(vitals);
35
36             double urgencyLambda = result.getProbability();
37             FogDevice localHw = getDeviceById(tuple.getSourceDeviceId());
38             FogDevice cloud = getDeviceByName("cloud");
39 }
```

F final ▾ main ▾

final ▾ PTSRAController.java

Project JDK is not defined

Setup SDK

```
40     double timeHw = estimateProcessingTimeHw(tuple, localHw, urgencyLambda);
41     double timeCloud = estimateProcessingTimeCloud(tuple, localHw, cloud, urgencyLambda);
42
43     FogDevice targetDevice;
44     if (urgencyLambda > 0.5 || timeHw < timeCloud) {
45         Logger.debug("PTS-RA", "Scheduling locally, Λ=" + urgencyLambda);
46         targetDevice = localHw;
47     } else {
48         Logger.debug("PTS-RA", "Offloading to cloud, Λ=" + urgencyLambda);
49         targetDevice = cloud;
50     }
51
52     updateQueueLength(targetDevice.getId(), 1);
53     tuple.setDestinationId(targetDevice.getId());
54     sendTupleToPlacement(tuple);
55 }
56
57
58 private double estimateProcessingTimeHw(Tuple tuple, FogDevice hw, double urgencyLambda) {
59     int queueLength = deviceQueueLengths.get(hw.getId());
60     double queuingTime = queueLength * 0.05;
61     double computationTime = tuple.getLength() / hw.getHost().getTotalMips();
62     return (queuingTime / urgencyLambda) + computationTime;
63 }
64
65 private double estimateProcessingTimeCloud(Tuple tuple, FogDevice hw, FogDevice cloud, double urgencyLambda) {
66     double transmissionTime = tuple.getLength() / hw.getUplinkBandwidth();
67     double computationTime = tuple.getLength() / cloud.getHost().getTotalMips();
68     return (transmissionTime / urgencyLambda) + computationTime;
69 }
70
71 private void updateQueueLength(int deviceId, int change) {
72     int currentLength = deviceQueueLengths.get(deviceId);
73     deviceQueueLengths.put(deviceId, currentLength + change);
74 }
```

package org.fog.ptsla;

F final main

PredictionResult.java PTSRAController.java

Project JDK is not defined

Setup SDK

```
60     double queuingTime = queueLength * 0.05;
61     double computationTime = tuple.getLength() / hw.getHost().getTotalMips();
62     return (queuingTime / urgencyLambda) + computationTime;
63 }
64
65 private double estimateProcessingTimeCloud(Tuple tuple, FogDevice hw, FogDevice cloud, double urgencyLambda) {
66     double transmissionTime = tuple.getLength() / hw.getUplinkBandwidth();
67     double computationTime = tuple.getLength() / cloud.getHost().getTotalMips();
68     return (transmissionTime / urgencyLambda) + computationTime;
69 }
70
71 private void updateQueueLength(int deviceId, int change) {
72     int currentLength = deviceQueueLengths.get(deviceId);
73     deviceQueueLengths.put(deviceId, currentLength + change);
74 }
75
76 private FogDevice getDeviceByName(String name) {
77     for (FogDevice device : getFogDevices()) {
78         if (device.getName().equals(name)) return device;
79     }
80     return null;
81 }
82
83 private FogDevice getDeviceById(int id) {
84     for (FogDevice device : getFogDevices()) {
85         if (device.getId() == id) return device;
86     }
87     return null;
88 }
89 }
```

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

```
1 package org.fog.ptsra;
2
3 > import ...
...
6
7 public class SensorDataReader {
8
9     public static List<SensorData> readSensorData(String filePath) {
10         List<SensorData> sensorDataList = new ArrayList<>();
11
12         try (BufferedReader br = new BufferedReader(new FileReader(filePath))) {
13             String line;
14             boolean firstLine = true;
15
16             while ((line = br.readLine()) != null) {
17                 if (firstLine) { firstLine = false; continue; }
18
19                 String[] values = line.split(",");
20                 if (values.length >= 4) {
21                     SensorData data = new SensorData();
22                     data.patientId = values[0].trim();
23                     data.heartRate = Double.parseDouble(values[1].trim());
24                     data.bloodPressure = Double.parseDouble(values[2].trim());
25                     data.glucoseLevel = Double.parseDouble(values[3].trim());
26                     sensorDataList.add(data);
27                 }
28             }
29             Log.println("Read " + sensorDataList.size() + " sensor data records");
30
31         } catch (Exception e) {
32             Log.println("Error reading sensor data: " + e.getMessage());
33         }
34         return sensorDataList;
35     }
36
37     public static class SensorData {
38         public String patientId;
```

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



```
1 package org.ptsra;
2
3 > import ...
...
8
9 public class PythonModelClient {
10
11     private static final String API_URL = "http://127.0.0.1:5000/predict"; // Flask API endpoint
12
13     public static PredictionResult getPrediction(Map<String, Double> vitals) {
14         try {
15             // Convert vitals map to JSON
16             ObjectMapper mapper = new ObjectMapper();
17             String jsonInput = mapper.writeValueAsString(vitals);
18
19             // Create connection
20             URL url = new URL(API_URL);
21             HttpURLConnection conn = (HttpURLConnection) url.openConnection();
22             conn.setRequestMethod("POST");
23             conn.setRequestProperty("Content-Type", "application/json");
24             conn.setDoOutput(true);
25
26             // Send request
27             try (OutputStream os = conn.getOutputStream()) {
28                 byte[] input = jsonInput.getBytes("utf-8");
29                 os.write(input, 0, input.length);
30             }
31
32             // Read response
33             StringBuilder response;
34             try (BufferedReader br = new BufferedReader(new InputStreamReader(conn.getInputStream(), "utf-8"))) {
35                 response = new StringBuilder();
36                 String responseLine;
37                 while ((responseLine = br.readLine()) != null) {
38                     response.append(responseLine.trim());
39                 }
40             }
41         } catch (IOException e) {
42             e.printStackTrace();
43         }
44     }
45 }
```



PredictionResult.java

PTSRAController.java

SensorDataReader.java

PythonModelClient.java

Setup SDK

⚠ Project JDK is not defined

```
26 // Send request
27 try (OutputStream os = conn.getOutputStream()) {
28     byte[] input = jsonInput.getBytes("utf-8");
29     os.write(input, 0, input.length);
30 }
31
32 // Read response
33 StringBuilder response;
34 try (BufferedReader br = new BufferedReader(new InputStreamReader(conn.getInputStream(), "utf-8"))) {
35     response = new StringBuilder();
36     String responseLine;
37     while ((responseLine = br.readLine()) != null) {
38         response.append(responseLine.trim());
39     }
40 }
41
42 // Parse JSON response
43 Map<String, Object> respMap = mapper.readValue(response.toString(), Map.class);
44 double prob = ((Number) respMap.get("probability")).doubleValue();
45 int pred = ((Number) respMap.get("pred")).intValue();
46 String label = (String) respMap.get("label");
47
48     return new PredictionResult(prob, pred, label);
49
50 } catch (Exception e) {
51     e.printStackTrace();
52     return new PredictionResult(0.0, 0, "Error");
53 }
54 }
55 }
56 }
```

OFF

```
> final > app.py > ...
```

```
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

Data Reception

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

Security Preprocessing via CI

All new data/tasks are first passed through the CI 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.

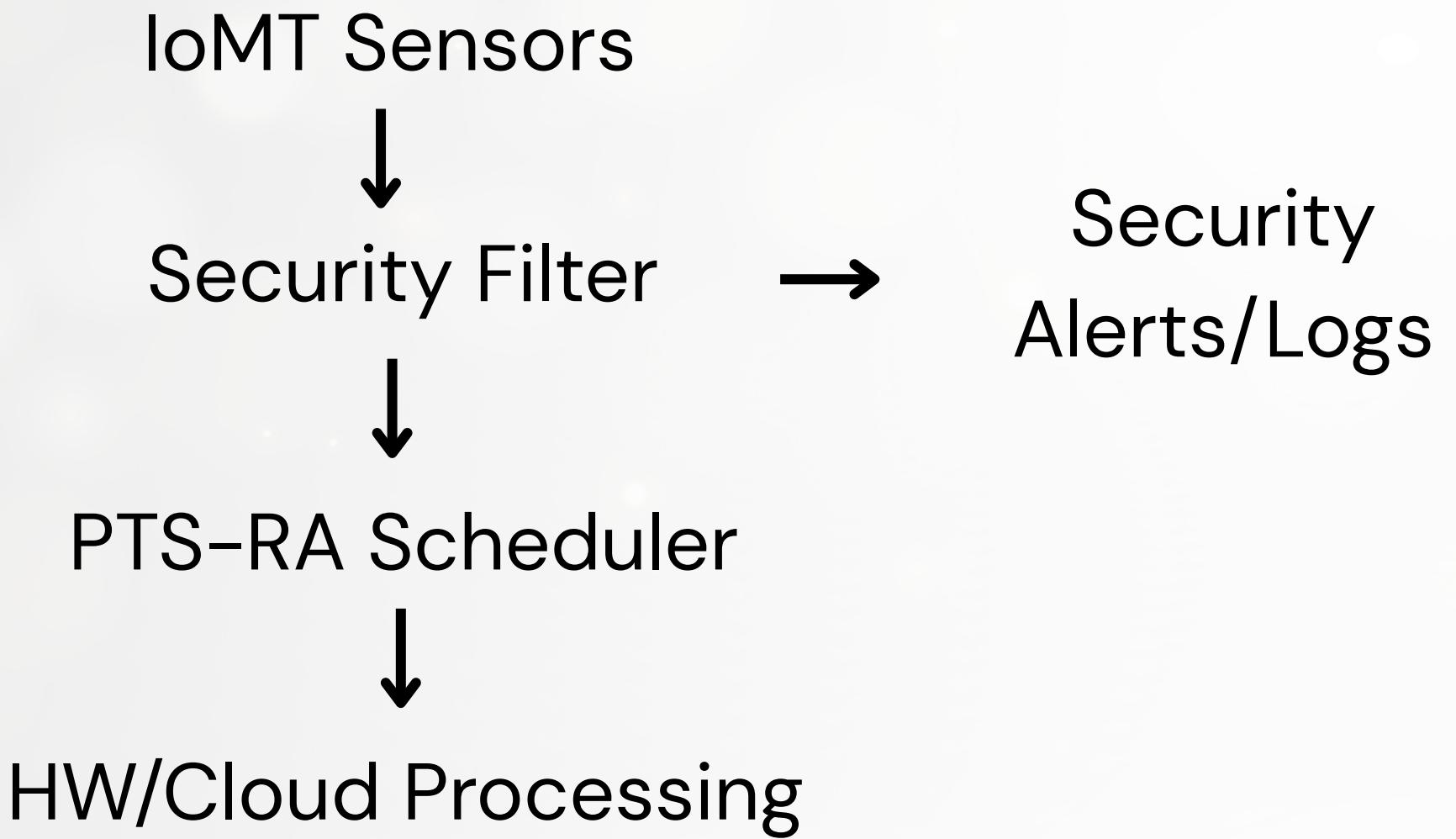
If clean, the task moves forward to priority-based scheduling.

Secure Task Processing

Only tasks verified by CI 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.

How CI Works for Edge Scheduling



Implementation Architecture

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

Low Overhead: Designed for edge deployment –CNN runs in real time with minimal latency added to each task, keeping overall emergency response rapid.

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

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

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:
- $X = [\text{HR}, \text{SpO}_2, \text{RR}, \text{Temp}, \text{SBP}, \text{DBP}, \text{HRV}, \text{WinMeanHR}]$

Step 3: Thresholding & Priority Assignment

- DBN outputs urgency probability for each patient/task.
- Thresholding decides:
 - Above threshold → Urgent
 - Below threshold → Non-Urgent
- Urgent tasks get priority scores for scheduling at the edge.

How CI Works for Edge Scheduling

Advantages

- Automated hyperparameter tuning via BFOA → higher F1 and balanced performance.
- Handles class imbalance using weighted loss and SMOTE oversampling.
- Captures complex temporal + physiological patterns from patient data.
- Improves scheduling decisions by accurately distinguishing urgent vs non-urgent tasks.

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

2

3

Context Awareness:

Integrates both patient health vitals (HR, SpO₂, Temp, BP, etc.) and system resource availability (CPU, bandwidth, storage) to make balanced scheduling decisions.

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.

Real-Time Adaptation:

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

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

**CI-Enhanced
Edge Benefits for
Healthcare**



QUESTIONS?