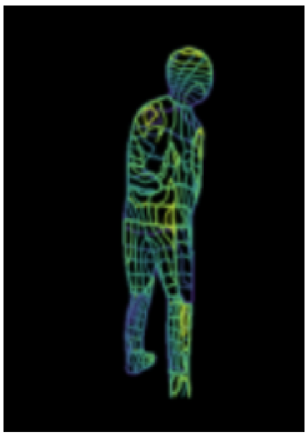
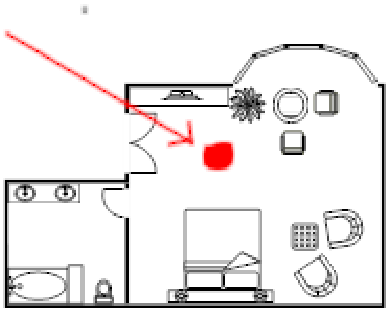
CPS Project - WiSentinel (Main page)

A **configurable** cyber-physical system that detects unauthorized presence/poses through/inside walls using WiFi signals, alerting users via Telegram or Email with **anonymized silhouettes** and **coordinates on a floor plan**—no cameras, no wearables.
L rules

Minimum Technical Requirements (MVP Scope)

Functional Requirements (FR)

These are features the system **must perform**.

1) The system shall detect human presence based on WiFi CSI signals.
2) The system shall classify human pose into **at least 6 categories**: `standing`, `crouching`, `no person (empty room)`, `lying`, `T-Pose`.
3) The system shall localize the person in **one physical room**, mapped to a zone name (e.g., "Living Room").
4) The system shall trigger alerts based on **predefined pose + time + zone rules** (e.g., crouching after 10 PM).
5) The system shall send real-time alerts via **Telegram Bot (or email)**, including pose label, zone, timestamp, and silhouette image.
6) The system shall log all detections and alerts in a local file (`.json` or `.csv`) for evaluation purposes.
7) The system shall support manual ground truth labeling using a **single RGB camera and OpenPose** for training dataset creation.

Non-Functional Requirements (NFR)

These define **quality attributes** and constraints on how the system performs.

1) The system shall maintain **latency below 5 seconds** from detection to alert for real-time usability.

2) The system shall be deployable on a single laptop or local PC with no cloud dependencies.
3) All components (data capture, ML inference, alert engine) shall be **modular**, allowing later replacement or upgrades.
4) The system shall work reliably in **low-light or no-light conditions**, without vision-based inference at runtime.

Project Constraints ©

These are **limits or boundaries** imposed on the project due to time, hardware, or scope.

1) The system shall be deployed in one room only (no multi-room tracking).
2) The system must support **at least one person at a time** (single-subject detection).
3) CSI data must be collected using **TP-Link Archer A7 routers** with OpenWRT and Nexmon CSI plugin.

Roles in the project

This table defines **who does what** in the WiSentinel project, broken down by responsibilities and collaboration links.

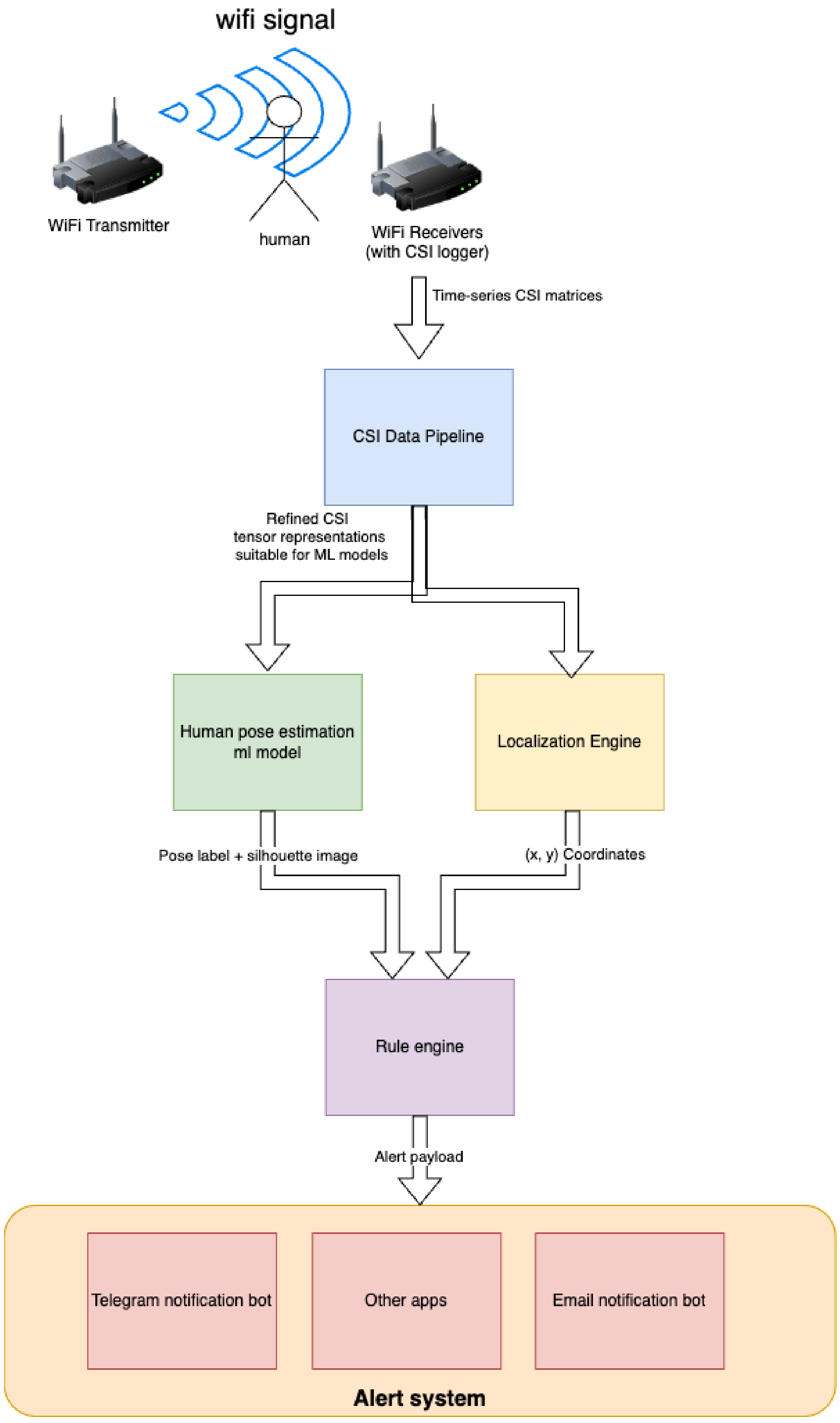
Develops: The **core component** you're building. This matches a major block in the system architecture (e.g., CSI pipeline, Pose Estimation, Rule Engine).
Responsibilities: A breakdown of **your tasks** — what you're expected to implement, build, or contribute. These are active, visible outputs you'll develop over time.
Collaborates With: People you should coordinate with frequently. This is based on data dependencies or output connections (e.g., Pose Estimation outputs are used in Rules and Alerts).
Obviously members in the **same color** collaborate with each other as well.

orange - **CSI & Hardware:** Responsible for signal capture, preprocessing, and infrastructure.
green - **ML & Logic Layer:** Focused on learning models, decision logic, and user-facing alerts.

Name	**Develops**	**Responsibilities**	**Collaborates With**
Ilya	- Hardware - CSI and Data pipeline - Architecture	- Hardware setup - CSI data capture - Define architecture and system flow - Integrate all components - Coordinate roadmap - Help other team members	- Max - Lam - Shivam
Lam	CSI Tooling & Feature Prep (CSI and Data pipeline)	- Implement CSI denoising/filtering - Build plotting and slicing tools - Create test-ready datasets	- Ilya - Rose - Max
Shivam	Localization Engine	- Implement RSSI or CSI-based localization	- Ilya - Lam

		- Map (x, y) to zone - Draw overlay on floor plan	- Rose
Max	Pose Estimation ML Model	- Train pose model on CSI - Build pose model pipeline - Collaborate on camera labeling prototype	- Ilya - Lam
Rose	Rule Engine & Alert System	- YAML-based rule engine - Telegram/email bot alerts - Collaborate on ML model output integration	- Max - Shivam

Architecture



WiFi Transmitter

What it is: A TP-Link A7 router configured to broadcast WiFi signals.
Function: Continuously sends packets that propagate and reflect through the room, enabling CSI collection. Can be achieved by using **ping** utility.
Technologies: TP-Link A7 v5 with a custom OS called OpenWRT**.** OpenWRT is based on linux.
Input: None (initiates transmission)
Notes: Use 5 GHz band for finer CSI resolution; should support stable broadcasting on a fixed channel (e.g., ch. 36 or 128)

WiFi Receiver (with CSI logging)

What it is:A TP-Link A7 router with Nexmon or other CSI data collection utility.
Function: Capture CSI — complex amplitude and phase info per subcarrier.
Technologies: OpenWRT + Nexmon_CSI/atheros CSI
Input: WiFi signals sent by wifi transmitter. Depending on what absorbed/reflected the signal (metal, wood, flesh) the signal will be different.
Output: CSI samples as time-series data:

- Shape: (tx_ant, rx_ant, subcarriers, timesteps)
- Example: 3x3x30x20 complex matrix
    - 3x3 - 3 antennas on wifi transmitter and 3 antennas on receiver. Each transceiver antenna sends data to each receiver antenna.
    - 30 subcarriers
    - 20 timestamps
- **Notes**: Output is usually logged in **`.dat`** or **`.csv`** files with timestamps. But for real time app output will be streamed e.g. via udp socket.

CSI Data pipeline

What it is: Python application that cleans, normalizes, and extracts features from raw CSI.
Function: Removes phase noise, applies calibration, aggregates useful temporal patterns. Prepares data for ML model.
Technologies:

- Python (NumPy, SciPy, pandas)
- Jupyter for prototyping

Input: Raw Time Series CSI matrix (e.g., **`.csv`** or **`.npy`** `or` **`streamed`**)
Output: Normalized tensor or feature vector ready for ML:

- Shape: [timesteps, features] or [channels, height, width]
- Format: `.npy` or passed directly to model

Notes: May apply filtering, windowing, PCA, or FFT depending on model needs.

Human Pose Estimation ML Model

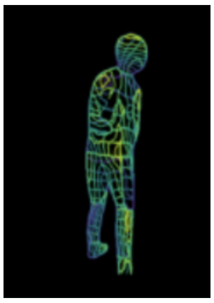
What it is: A neural network that classifies human pose from CSI input.
Function: Outputs pose label (`standing`, `running`, crouching, etc.) and optional silhouette image.

Technologies:

- PyTorch / TensorFlow / OpenCV
- RF-Pose, DensePose, or WiSPPN-like custom model

Input: Refined CSI tensor
Output:

- pose_label: string
- silhouette_img



Notes:

- May require transfer learning or domain adaptation due to RF-to-vision gap. Pretraining on RF datasets helps (WiSPPN, etc.).
- There are two approaches to this problem:
    - Use more popular pose estimation models from camera images and train pose estimation models based on CSI data and using camera image models as a teacher.
    - Find a model which builds silhouette image from CSI data (and additionally train it if needed). Then feed this image into a pose estimation model designed for camera images.

Localization Engine

What it is: Module using RSSI or CSI signal strength across receivers to estimate 2D position.
Function: Estimates coordinates using **trilateration**.
Technologies:

- Python (NumPy, Matplotlib, OpenCV)
- Optional: Kalman filter for smoothing

Input:

- RSSI or CSI-derived signal strength per router
- Geometry info (router positions)

Output:

- Coordinates: `(x, y)` in meters
- Optionally rendered on floorplan image

Notes:

- Output can be stored as dict or overlayed as an image with cv2
- Third router might be needed. Should be able to be replaced with a regular laptop.

Rule Engine

What it is: A YAML-based rules interpreter that triggers alerts based on logic specified in YAML config.
Function: Evaluates conditions like time, pose, zone → generates alert payload, sends get request to APIs of other applications compatible with our system.

Technologies:

- Python (PyYAML)
- FastAPI backend

Input (example):
Gets information from two ML model

Information **Human Pose estimation model** {
```
  "pose": "crouching",
  "time": "23:41",
  "zone": "Living Room",
  "confidence": 0.91,
}
```

and **Localization Engine**
```
"coordinates" {
     "X" : 115,
     "Y" : 100
}
```

Output (example):
Triggers alert in alerting system app (or any compatible app) i.e. sends GET request to app's url.

```
{
  "alert": true,
  "payload": {
    "pose": "crouching",
    "zone": "Living Room",
    "timestamp": "...",
    "image_path": "silhouette.png"
  }
}
```

Notes: Logic for rule is stored in YAML (`rules.yaml`), examples:

```
- condition: "time > '22:00' and pose == 'crouching'"
  action: "send_alert"
```

Alert System

Set of applications that have an API compatible with the **Rule Engine.**

Telegram notification bot

What it is: A Python bot using Telegram Bot API.
Function: Sends formatted messages + image overlays to the user.
Technologies: python-telegram-bot or telebot

Input: Alert payload (pose, zone, time, image)
Output: Telegram message:
🚨 Intrusion Detected!
• Pose: Crouching
• Time: 23:41
• Zone: Living Room
[silhouette image]

Email Notification Bot

What it is: SMTP/SendGrid-powered email notifier.
Function: Sends alert summaries via email.
Technologies: Python **smtplib** or **sendgrid** SDK
Input: Same as Telegram payload
Output: HTML or plaintext email with alert details

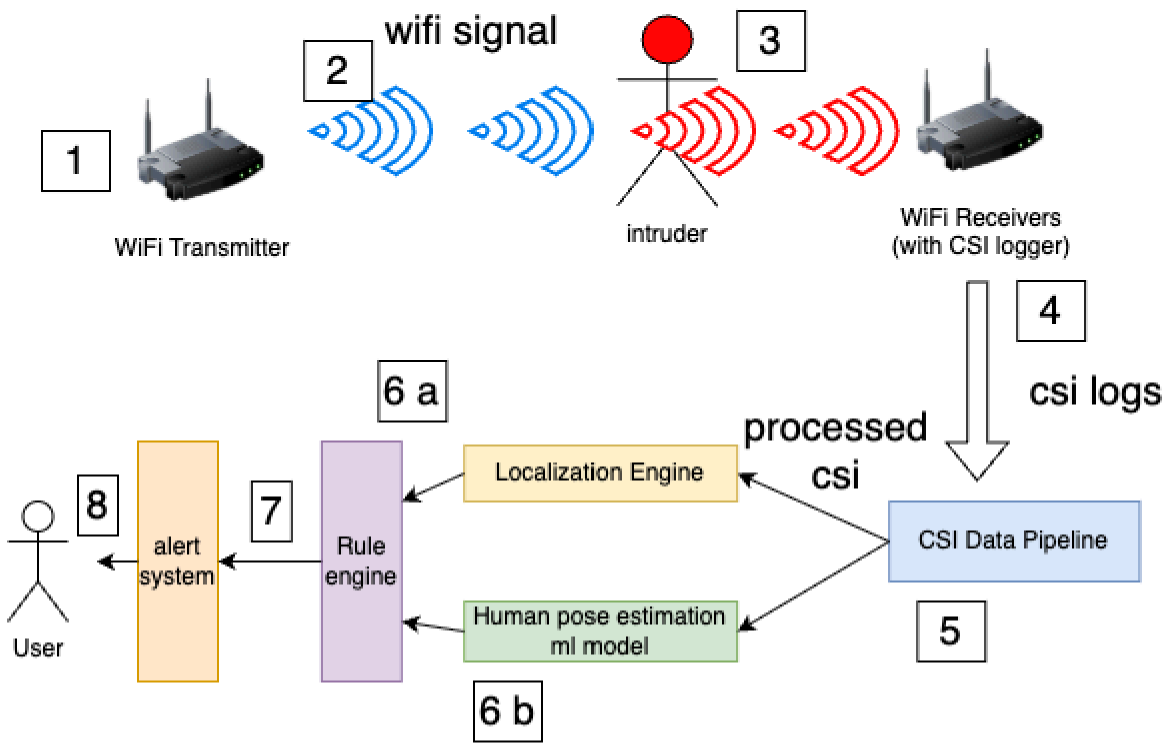
Other Apps

Function: Could include:

- Logging service (to DB)
- Web dashboard (FastAPI + JS)
- Smart home triggers (e.g., Home Assistant integration)

USE CASE Example

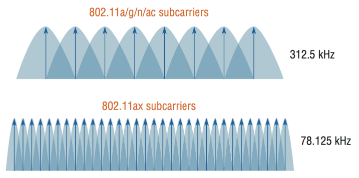
The User sets up **WiSentinel** to monitor their home during the night. While asleep, an **unauthorized person** enters the living room through the backyard door. WiSentinel **detects** their presence, estimates that the person is **crouching**, and immediately **sends** a privacy-preserving Telegram **alert** to the resident's phone with location on a floor plan of **User's** home.



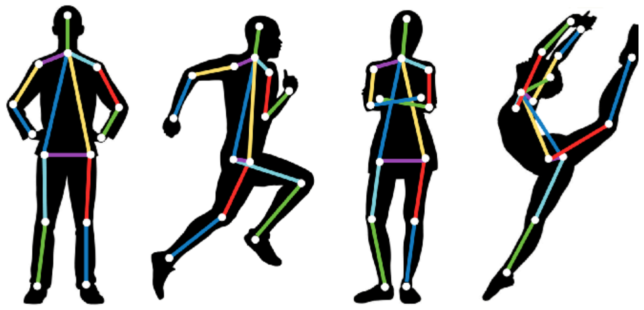
1. **Initialization**: System armed for monitoring.
2. **Transmission**: Wifi transmitter continuously broadcasts WiFi waves into space
3. **Reflection**: Person disturbs signal pattern causing several effects like **absorption**, **reflection**, **scattering**, **attenuation**. The patterns of these signal changes can be used to build a mapping to human poses.
4. **Capture**: CSI logs generated and streamed for processing.
5. **Processing**: CSI preprocessed into usable format (noise removal, perhaps FFT).
6. **Inference**:
    a. Location estimated: x = 100, y = 150
    b. Pose classified: crouching
7. **Evaluation**: Rule engine matches conditions:
   "time > '22:00' and pose == 'crouching' and location == (100,150)"
8. **Notification**: Alert sent to **user's** phone with **silhouette** and **location** on floor plan.

Main terms and definitions

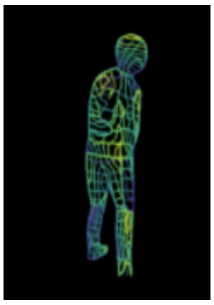
- **CSI (Channel State Information)** - Fine-grained physical layer data extracted from WiFi signals. It captures how each subcarrier (small frequency chunk) is affected by the environment. Used to detect presence and movement of humans by analyzing amplitude and phase shifts across time.



- **RSSI (Received Signal Strength Indicator)** - A simpler signal strength measurement, showing total power received from a WiFi signal. Used for estimating a person's **approximate location** through triangulation.
- **Preprocessing (CSI)** - Cleaning, filtering, and reshaping raw CSI data to prepare it for use in machine learning. Includes denoising, windowing, normalization, and slicing into tensors.
- **Pose Estimation** - A machine learning technique that determines the **position or posture** of a person (e.g., standing, crouching) based on input data (images or CSI). In WiSentinel, this is done from CSI without visual cameras.



- **Silhouette** - A simplified, non-identifying outline of a person's body pose generated by the pose estimation model. Sent in alerts to preserve privacy.



- **Nexmon CSI** - A modified firmware/plugin that enables some routers (like TP-Link Archer A7) to extract CSI data that's normally inaccessible. Required for CSI logging on embedded devices.
- **Rule engine** - The logic-based system that analyzes pose, location, and time to decide whether to send an alert (e.g., "Crouching after 10 PM in Living Room"). Defined using YAML rules.
- **Localization** - Estimating a person's (x, y) position in a room using signal properties like RSSI. Mapped to user-defined zones (e.g., "Kitchen", "Backyard").
- **Fastapi** - A lightweight Python web framework used for building APIs. In WiSentinel, it can serve rule-based alerts or control interfaces.
- **Telegram Bot** - A Python-controlled bot that sends real-time notifications, pose images, and zone data directly to the user's phone.
- **Ground Truth Labeling** - The process of collecting correct pose information (from a superior camera-based system like OpenPose) to label CSI data during training
- **(x, y) Zone Mapping** - Translating numeric coordinates from localization into named zones (e.g., mapping x=2.1, y=3.4 → "Living Room"). Used in alert messages.

Materials and sources

- Person-in-WiFi 3D: End-to-End Multi-Person 3D Pose Estimation with Wi-Fi - https://openaccess.thecvf.com/content/CVPR2024/papers/Yan_Person-in-WiFi_3D_End-to-End_Multi-Person_3D_Pose_Estimation_with_Wi-Fi_CVPR_2024_paper.pdf
- DensePose From WiFi - https://arxiv.org/pdf/2301.00250
- Hands on guide on wireless sensing - https://tns.thss.tsinghua.edu.cn/~guoxuan/assets/pdf/Paper-Hands-On.pdf