****

**Cat Detector**

**Monitoring Front Door Area Using PIR Triggered Video Capture**

**and Machine Learning Classification**

**IoT & Edge Computing Project**

**Course: 4510 H4 DAKP**

**Supervisor: David Svarrer**

**Name: Tianzuo Wang**

**Date: 14 January 2026**

1. **Introduction**

Several cats visit my garden regularly; they tend to walk by or stay at the front door a bit. One particular cat, “Hana”, visits very frequently. This project addresses a practical curiosity: how often Hana visits.

This Project combines three technologies: motion detection, video capture and machine learning classification.

**An Arduino microcontroller** with two **PIR sensors** detect movement in the front door area and publish a trigger event via MQTT to a could broker. **A Raspberry Pi** subscribes to this trigger topic and records a 5-second video clip upon each event. Selected frames from the video are analyzed using **a TensorFlow Lite model** trained to distinguish Hana from other cats based on coat patterns. Each event is stored in a local SQLite database (analyzing result, confidence score, timestamp, sensor information, video file path, sensor location) and displayed on a Raspberry PI hosted dashboard, which is remotely accessible via **Tailscale VPN**.

By combining these components, this project demonstrates an event-driven edge computing pipeline: sensor-triggered capture, on-device ML inference, and local storage and visualization. Instead of uploading all media to the cloud, processing is performed locally on the Raspberry Pi close to the data source. This reduces bandwidth and highlights a decentralized data processing approach.

1. **Deployment Context and Constraints**

**2.1 Deployment Environment**



The detection is taking place in the narrow stone walkway between the main house and a storage shed. The walkway leads directly to the front door and is a common path for cats and people. The shed has a window facing the front door, which makes it an ideal observation point for monitoring activities in the front door area while keeping electronics sheltered.

This physical layout matters, because it determines the camera viewing angle and the distance to the target.

**2.2 Physical Constraints**

* **Outdoor weather exposure:**

The PIR sensors are mounted on the shed’s exterior wooden wall (enclosed in cardboard boxes for basic shielding). Mounting stability and long-term sensor protection is affected by weather conditions such as rain, snow, wind and humidity.

* **Glass barrier for PI camera:**

The PI camera records through window glass, which reduces image quality due to reflections (e.g. indicator LEDs from the Arduino and Raspberry PI) and lower contrast.

During rain and snow, water droplets and condensation on the glass can cause significant blur in the captured footage.

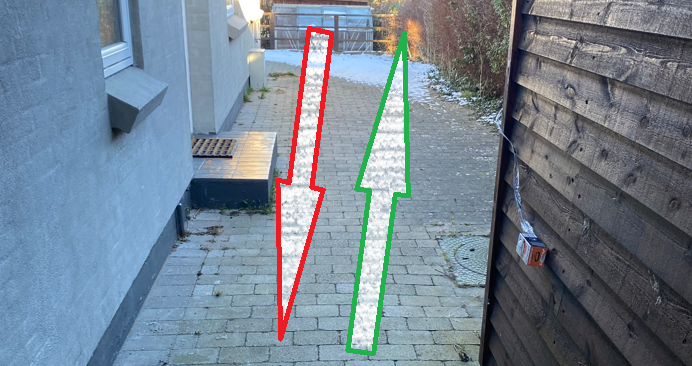




* **Placement of Arduino and PIR sensor**

Cats are small and move close to the ground. With the current setup, the PIR sensors mainly trigger when a cate moves **toward the front** door (green direction).

If a cat passes the front door first, which means entering and leaving the camera’s field of view, then triggers the sensor (red direction), the camera will capture an empty video without cat. Therefore, the classifier will produce a “no cat” prediction, even though a cat was present earlier.

****

* **Weak Wi-Fi signal inside the shed**

Wi-Fi coverage is weaker in the shed. The Arduino node is particularly affected due to its less robust Wi-Fi module, leading to frequent disconnections.

* 1. **Technical Constraints**
* **PI camera v1.3 not IR, nearly zero performance in low light**

The Pi Camera v1.3 has no IR night-vision capability. In low light, footage becomes too dark for reliable classification without additional lighting



* **Video Format and Browser Compatibility**

The camera produces raw .h264 recordings and require convention to .mp4 for browser playback

* **Limited training data (243 images)**

The machine learning model was trained on 151 Hana photos and 82 no\_Hana dataset, this is small for a machine learning dataset.

1. **Project Goals, Scop and Limitation**

**3.1 Objectives**

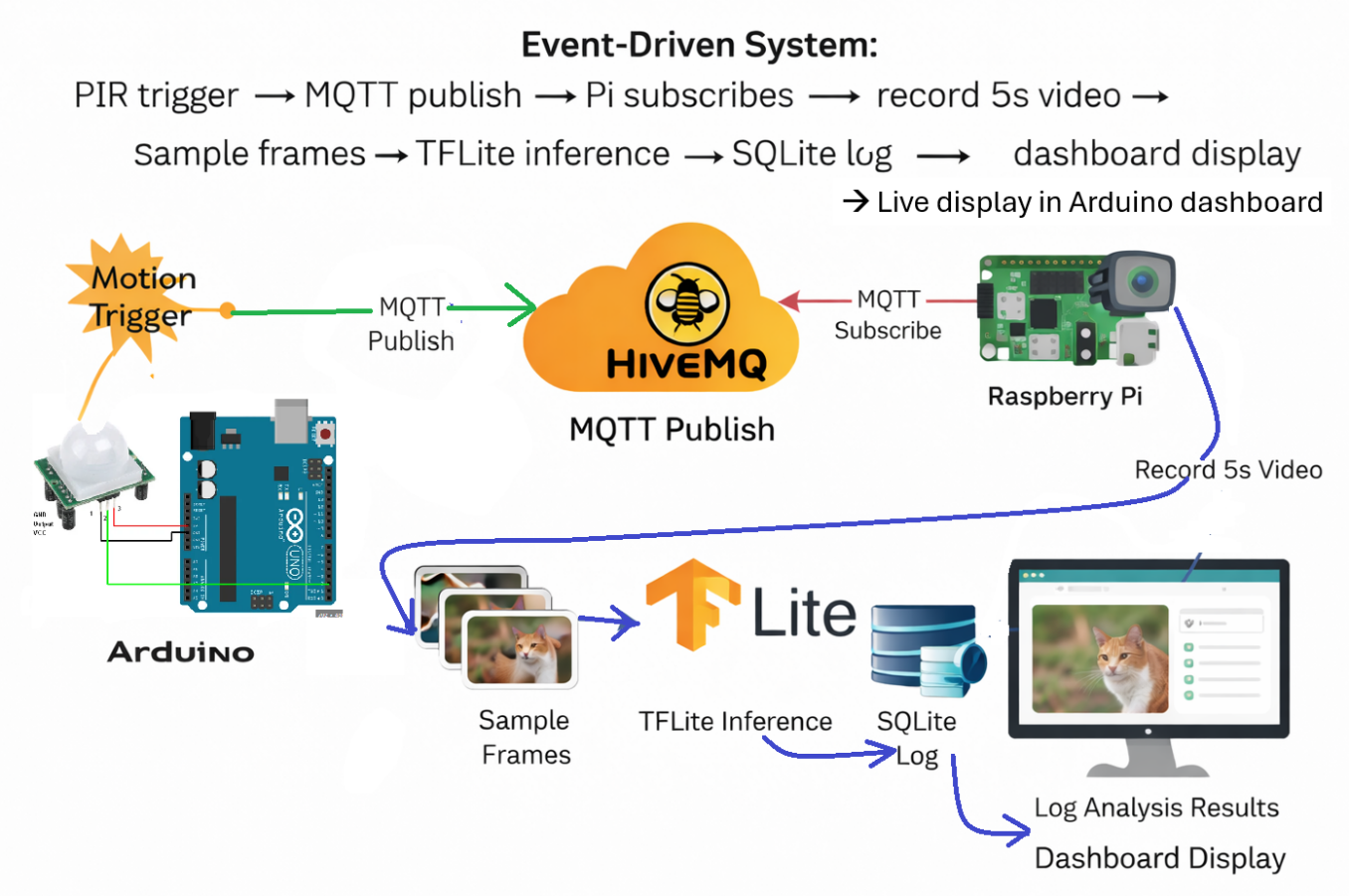
* Motion Detection: Detect movements near the front door reliably
* Capture video: Record a 5-second video upon each trigger of the PIR sensor(s)
* Analyze the video footage and classify if Hana is present with a confident score
* Store results in a database
* Showing server status, Arduino status and analyzing results along with videos playback on a dashboard.
* Remote access dashboard privately via Tailscale VPN

**3.2 Scope and Limitation**

* Use Pi camera v1.3 instead of Arduino camera module for better photo quality
* The Raspberry Pi and Arduino will be placed inside the shed to protect the electronics against winter weather conditions
* No night-time detection capability (due to lack of extra lighting/IR hardware)
* Model training data limited to available photos
* Movement-based triggering only

1. **System Design and Data Flow**

**4.1 Overview**



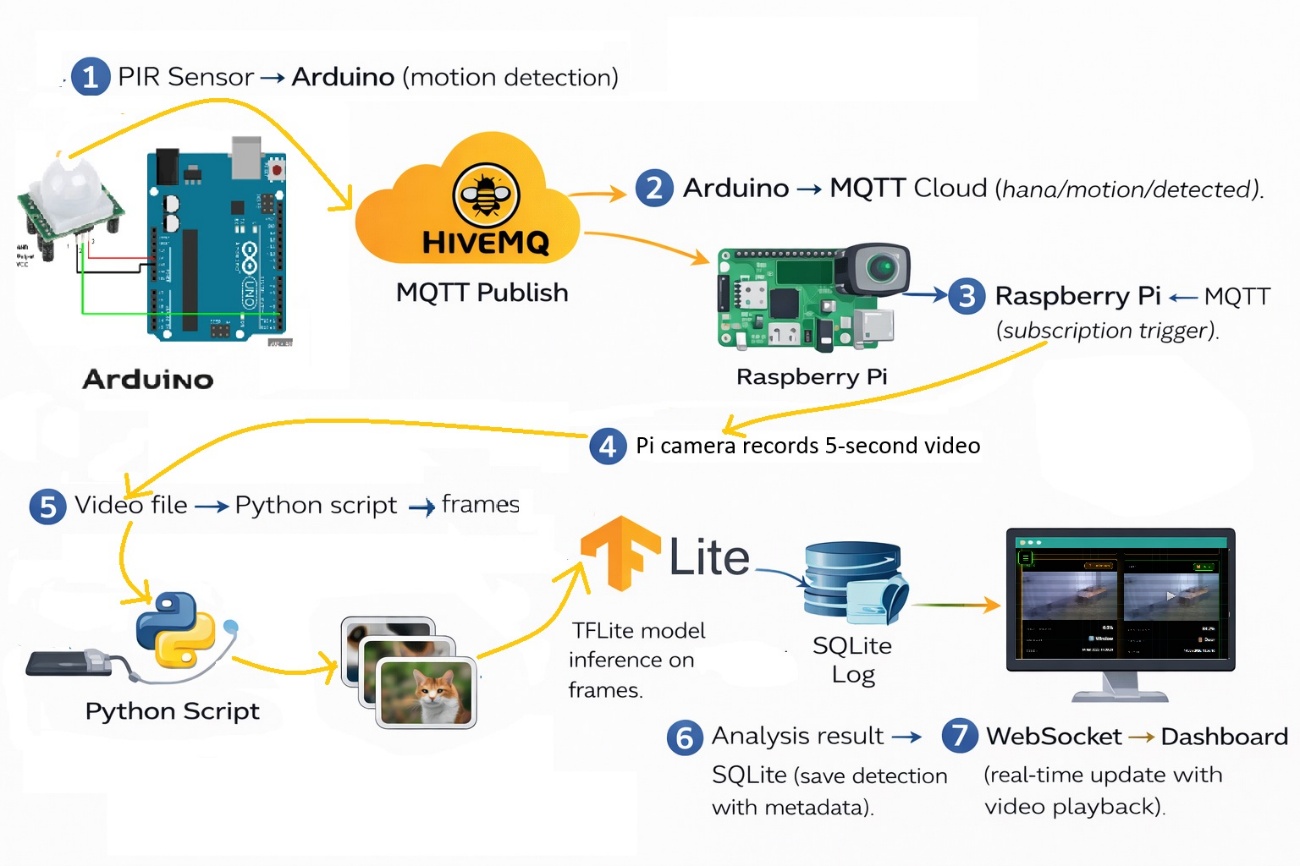
**4.2 Hardware Components**

* Arduino R4 WiFi
* Two PIR motion sensors
* Raspberry Pi 4 (server + inference host)
* Pi Camera v1.3
* Wi-Fi extender (TP-Link RE305)

**4.3 Software Components**

* MQTT publisher (Arduino)
* MQTT subscriber + capture pipeline (Raspberry Pi)
* Video conversion (FFmpeg: .h264 🡪 .mp4 for browser playback)
* TensorFlow Lite inference service
* SQLite database
* PM2 Process Manager (keeps the dashboard running, auto-restart, optional startup-on-boot)
* Tailscale VPN for private remote access
* Nginx serves the web dashboard and static media
* Ngrok used for GitHub webhook auto-deploy.

**4.4 Data Flow:**

**Communication Protocols:**

* MQTT: For Arduino-to Pi messaging
* WebSocket: for Pi-to-Dashboards real-time updates
* REST API: for requiring data from database for historical records

1. **Implementation**

**5.1 Initial setup of Raspberry Pi**

The Raspberry Pi 4 serves as the central processing unit, hosting the Node.js server, MQTT subscriber, ML inference service, and web dashboard.

**5.1.1 Installation of Ubuntu Server 24.04 LTS:**

* Download and install Raspberry Pi Imager
* Select **Other general-purpose OS**→ **Ubuntu Server** (64-bit LTS)
* Choose storage (SD card)
* Click settings icon to preconfigure:

Hostname: TianServer01

Username: tian`/ Password: (set the password)

**Enable SSH**

Configure Wi-Fi (SSID + password)

* Write image → Insert SD into Pi → Power on

**5.1.2 Initial SSH Connection**

Find Pi IP address, then connect from Windows PowerShell:

ssh tian@10.61.3.188

First-time connection prompts:

Are you sure you want to continue connecting? → Type yes

Enter password (may be forced to change on first login)

Successfully connected when prompt shows: tian@TianServer01:~$

**5.1.3 System Update and Security**

Update system packages:

sudo apt update && sudo apt upgrade -y

Enable automatic security updates:

sudo apt install -y unattended-upgrades

sudo dpkg-reconfigure --priority=low unattended-upgrades

**5.1.4 SSH Key Authentication Setup**

**5.1.4.1 Generate SSH key pair on Windows (PowerShell):**

# Create new Ed25519 key with passphrase

ssh-keygen -t ed25519 -a 64 -f $env:USERPROFILE\.ssh\id\_ed25519\_pi -C "tian@pi"

Enter a strong passphrase when prompted (protects private key)

Creates two files:

  Private key: C:\Users\<you>\.ssh\id\_ed25519\_pi (keep secret!)

  Public key: C:\Users\<you>\.ssh\id\_ed25519\_pi.pu (safe to copy)

**5.1.4.2 Copy public key to Pi:**

View public key

type $env:USERPROFILE\.ssh\id\_ed25519\_pi.pub

Copy entire line starting with "ssh-ed25519..."

**On Pi:**

# Create SSH directory and authorized\_keys file

mkdir -p ~/.ssh

nano ~/.ssh/authorized\_keys

# Paste public key on a new line

# Save: Ctrl+O, Enter

# Exit: Ctrl+X

# Set correct permissions (required for security)

chmod 700 ~/.ssh

chmod 600 ~/.ssh/authorized\_keys

**5.1.4.3 Test key-based login from Windows:**

# Login using specific key (will ask for passphrase, not Pi password)

ssh -i $env:USERPROFILE\.ssh\id\_ed25519\_pi tian@10.61.3.188

**5.1.4.4 SSH Agent: Avoid Repeated Passphrase Entry**

Configure Windows ssh-agent to remember key passphrase:

# Open PowerShell as Administrator

Set-Service ssh-agent -StartupType Automatic

Start-Service ssh-agent

# Add key to agent (enter passphrase once)

ssh-add $env:USERPROFILE\.ssh\id\_ed25519\_pi

# Verify key is loaded

ssh-add -l

Now SSH connections no longer prompt for passphrase until Windows restart. After reboot, run `ssh-add` again to reload the key.

**5.2 Installation of Application Dependencies**

**5.2.1 Base runtimes, libraries, and system utilities required to run the application:**

# Install Node.js runtime (v18.x) for server and dashboard

curl -fsSL https://deb.nodesource.com/setup\_18.x | sudo -E bash -

sudo apt install -y nodejs

# Install Python runtime and ML dependencies

sudo apt install -y python3-pip python3-opencv

pip3 install tensorflow tflite-runtime numpy opencv-python

# Install FFmpeg (video conversion utility)

sudo apt install -y ffmpeg

# Install PM2 (Node.js process manager)

sudo npm install -g pm2

# Install Nginx (web server / reverse proxy)

sudo apt install -y nginx

# Install Git (for webhook auto-deployment)

sudo apt install -y git

**5.2.2 Essential instructions:**

**5.2.2.1 Tailscale VPN Configuration:**

# Install Tailscale for secure remote access

curl -fsSL https://tailscale.com/install.sh | sh

sudo tailscale up

# Enable Tailscale on boot

sudo systemctl enable tailscaled

**5.2.2.2 Nginx Configuration:**

Nginx serves the React dashboard and provides access to video files:

nginx

server {

    listen 80;

    server\_name localhost;

    # Serve React dashboard

    location / {

        proxy\_pass http://localhost:5173;

        proxy\_http\_version 1.1;

        proxy\_set\_header Upgrade $http\_upgrade;

        proxy\_set\_header Connection 'upgrade';

        proxy\_set\_header Host $host;

        proxy\_cache\_bypass $http\_upgrade;

    }

    # Serve video files

    location /videos/ {

        alias /home/tian/cat\_videos/;

        autoindex on;

    }

    # API endpoints

    location /api/ {

        proxy\_pass http://localhost:3000;

    }

}

**5.2.2.3 GitHub Auto-Deployment with Webhook:**

The webhook listener in server/routes/webhook.js automatically pulls latest changes and restarts services when code is pushed to GitHub.

# Install ngrok for webhook tunnel

wget https://bin.equinox.io/c/bNyj1mQVY4c/ngrok-v3-stable-linux-arm64.tgz

sudo tar xvzf ngrok-v3-stable-linux-arm64.tgz -C /usr/local/bin

# Configure ngrok authentication

ngrok config add-authtoken TOKEN\_HERE

# Start ngrok tunnel (run in background)

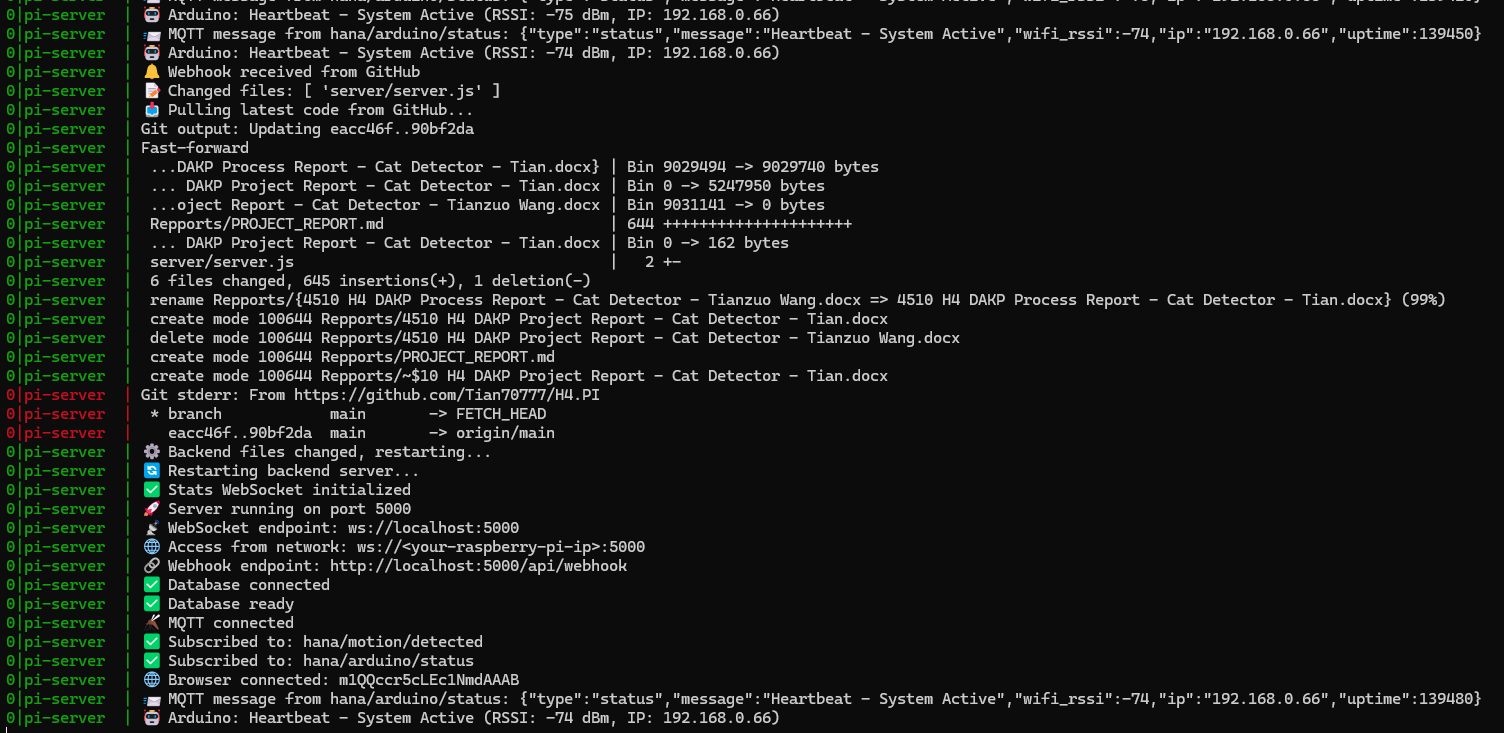
ngrok http 3000 &

# Add webhook endpoint to GitHub repository settings:

# Payload URL: https:// NGROK\_URL/webhook/github

# Content type: application/json

# Events: Just the push event



**5.2.2.4 PM2 Process Management:**

# Start server with PM2

cd /home/tian/H4.PI/server

pm2 start server.js --name cat-detector-server

# Start dashboard

cd /home/tian/H4.PI/dashboard

pm2 start "npm run dev" --name cat-detector-dashboard

# Save PM2 configuration

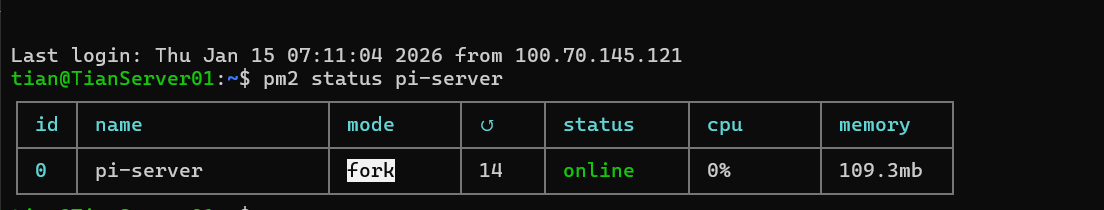
pm2 save

# Enable PM2 startup on boot

pm2 startup

sudo env PATH=$PATH:/usr/bin pm2 startup systemd -u tian --hp /home/tian

The webhook listener in server/routes/webhook.js automatically pulls latest changes and restarts services when code is pushed to GitHub.



**5.3 Hardware Components**

**5.3.1 Arduino UNO R4 Wi-Fi:**

* + Microcontroller with built-in Wi-Fi connectivity
  + Two GPIO pins (2, 4) connected to PIR sensors
  + Connected to network via Wi-Fi with TP-Link RE305 extender for improved signal
  + Configured with HiveMQ cloud MQTT credentials

**5.3.2 PIR Motion Sensors (HC-SR501):**

* + Two sensors positioned at different angles
  + Sensitivity adjusted via onboard potentiometer
  + Protected in cardboard box
  + 3-wire connection: VCC (5V), GND, OUT (digital signal)

**5.3.3 Raspberry Pi 4 (8GB):**

* + Runs server processes, MQTT client, and ML inference
  + Hosts SQLite database and video storage

**5.3.4 Pi Camera v1.3:**

* + Mounted inside shed window facing walkway
  + Records at 720p, 15fps for storage
  + Limited low-light performance (no IR capability)
    1. **TP-Link RE305 Wi-Fi Extender:**
  + Improves WiFi signal strength in shed area
  + Reduces Arduino disconnection frequency
  + Provides more stable MQTT connection

**5.4 Software Components**

**5.4.1 MQTT Publisher (Arduino):**

* + Firmware written in C++ using Arduino IDE
  + Libraries: `WiFiS3.h`, `ArduinoMqttClient.h`
  + Publishes to topics: `hana/motion/detected`, `hana/arduino/status`
  + Implements automatic WiFi reconnection with retry logic

**5.4.2 MQTT Subscriber (Node.js):**

* + `mqttService.js` connects to HiveMQ cloud broker
  + Subscribes to motion events and triggers video capture
  + Forwards Arduino status updates to dashboard via WebSocket

**5.4.3 Video Conversion (FFmpeg):**

* + Converts H.264 raw video to MP4 container
  + Command: `ffmpeg -i input.h264 -c copy output.mp4`
  + Preserves video codec (copy mode) for fast conversion
  + Required for browser playback compatibility
    1. **TensorFlow Lite Inference Service:**
  + Python script `cat\_detector\_cli.py` performs frame-by-frame analysis
  + Model: MobileNetV2-based CNN (4MB .tflite file)
  + Input: 224x224 RGB images, normalized to [-1, 1]
  + Output: Binary classification (hana/no\_hana) with confidence score
    1. **SQLite Database:**
  + Lightweight embedded database for detection logging
  + Schema includes: timestamp, classification result, confidence, sensor data, video path
  + Managed by `databaseService.js` with prepared statements
  + Database file: `/home/tian/H4.PI/data/detections.db`
    1. **Ngrok Tunnel:**
  + Exposes local webhook endpoint to GitHub
  + Required for GitHub webhook delivery to private
  + Provides HTTPS URL for webhook configuration
  + Used exclusively for auto-deployment

**Install ngrok:**

curl -s https://ngrok-agent.s3.amazonaws.com/ngrok.asc | sudo tee /etc/apt/trusted.gpg.d/ngrok.asc >/dev/null

echo "deb https://ngrok-agent.s3.amazonaws.com buster main" | sudo tee /etc/apt/sources.list.d/ngrok.list

sudo apt update && sudo apt install ngrok

# Get free account at https://dashboard.ngrok.com/get-started/setup

# Copy the **authtoken**, then:

ngrok config add-authtoken token

https://dashboard.ngrok.com/get-started/your-authtoken

# Start tunnel on port 5000

ngrok http 5000

**5.5 Motion detection (Arduino)**

The Arduino continuously monitors two PIR sensors (door sensor on pin 2, window sensor on pin 4). Upon detecting motion, it publishes a JSON message to the MQTT topic `hana/motion/detected`:

{

  "sensor1": true,

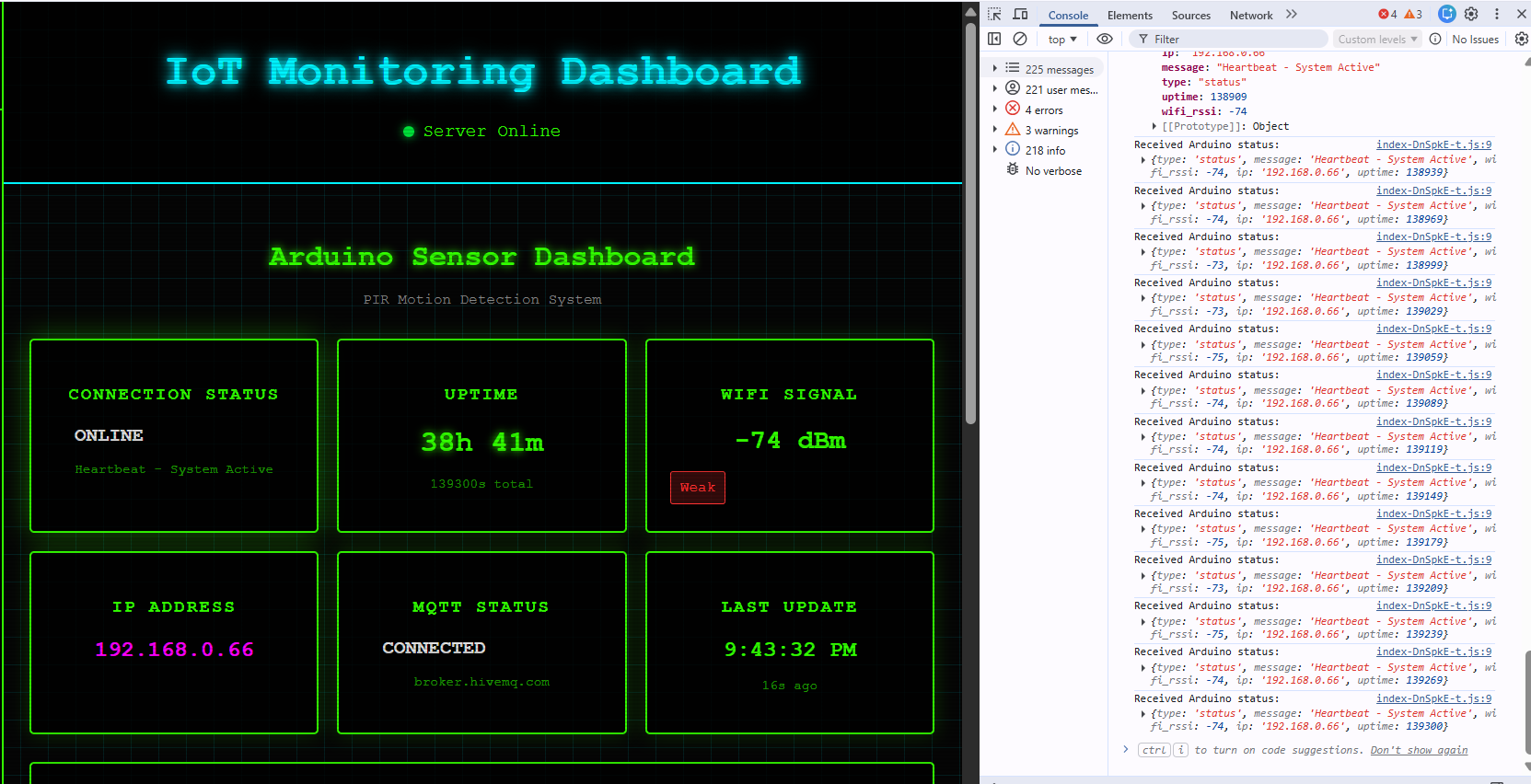
  "sensor2": false,

  "location": "sensor1",

  "timestamp": 1736345678

}

Arduino also publishes heartbeat status messages every 30 seconds to `hana/arduino/status` for remote monitoring, including Wi-Fi RSSI and IP address. The system implements automatic Wi-Fi reconnection with continuous retry logic.



**5.6 Video Capture and Processing (Raspberry Pi)**

Upon receiving an MQTT trigger, the Node.js `mqttService.js` initiates video capture through `cameraService.js`. The service uses `rpicam-vid` to record:

Resolution: 1280x720 (720p)

Duration: 5 seconds

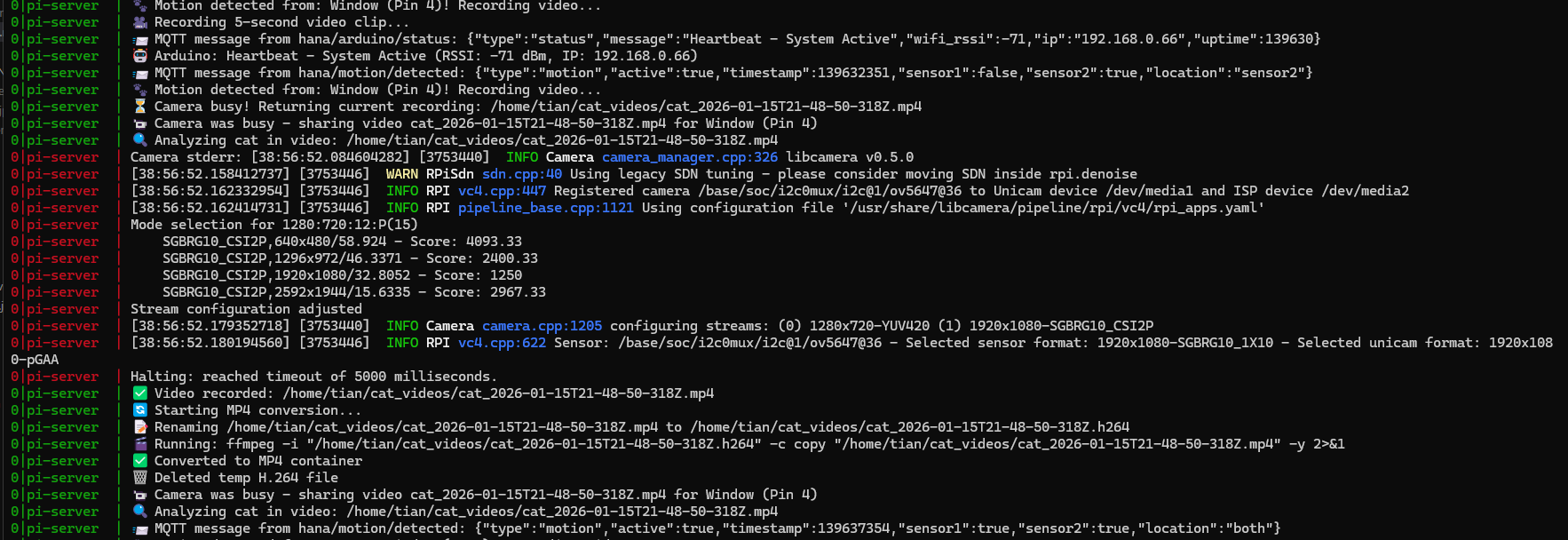
Framerate: 15 fps (storage-efficient)

Codec: H.264 (hardware-accelerated)

Videos are saved to `/home/tian/cat\_videos/` with ISO timestamp filenames. The service implements camera locking to prevent simultaneous recordings—if the camera is busy, subsequent triggers share the current recording.

Raw H.264 files are converted to MP4 containers using FFmpeg for browser compatibility:

ffmpeg -i input.h264 -c copy output.mp4



**5.7** **Machine Learning Classification**

The Python script `cat\_detector\_cli.py` analyzes videos using a TensorFlow Lite MobileNetV2-based model. The process:

* + **Frame sampling:** Extract frames at 15fps intervals
  + **Preprocessing:** Resize to model input size, normalize to [-1, 1]
  + **Inference:** Run TFLite interpreter on each frame
  + **Aggregation:** Apply 50% confidence threshold; classify video as "Hana" if any frame exceeds threshold

The model outputs:

{

  "class": "hana",

  "confidence": 0.87,

  "video\_analysis": {

    "hana\_percentage": 65.2,

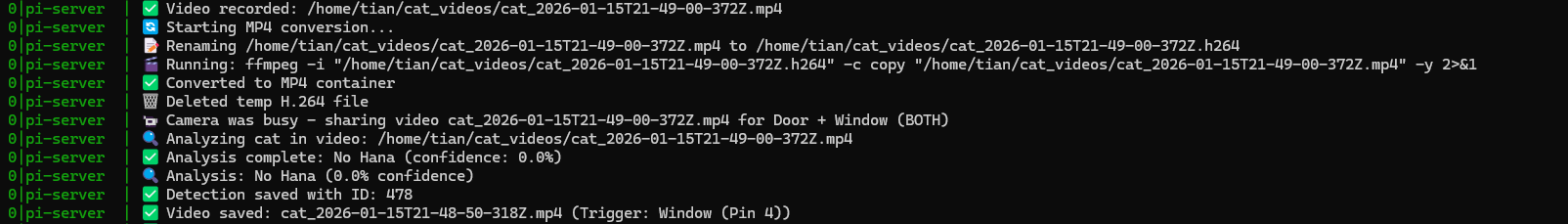
    "frames\_analyzed": 10,

    "duration": 5.0

  },

  "detection\_time\_ms": 1234.56

}



**5.8 Data Persistence**

The `databaseService.js` saves each detection to SQLite with the following schema:

**Table: `detections`**

- `id`: Auto-increment primary key

- `timestamp`: Detection time (ISO format)

- `is\_hana`: Boolean (1=Hana detected, 0=no Hana)

- `confidence`: Float (0-1)

- `photo\_path`: Video file path

- `sensor1`, `sensor2`: Which sensors triggered

- `location`: 'sensor1', 'sensor2', or 'both'

Database files are stored in `/home/tian/H4.PI/data/` and excluded from Git.

**5.9 Dashboard**

The React/TypeScript dashboard displays:

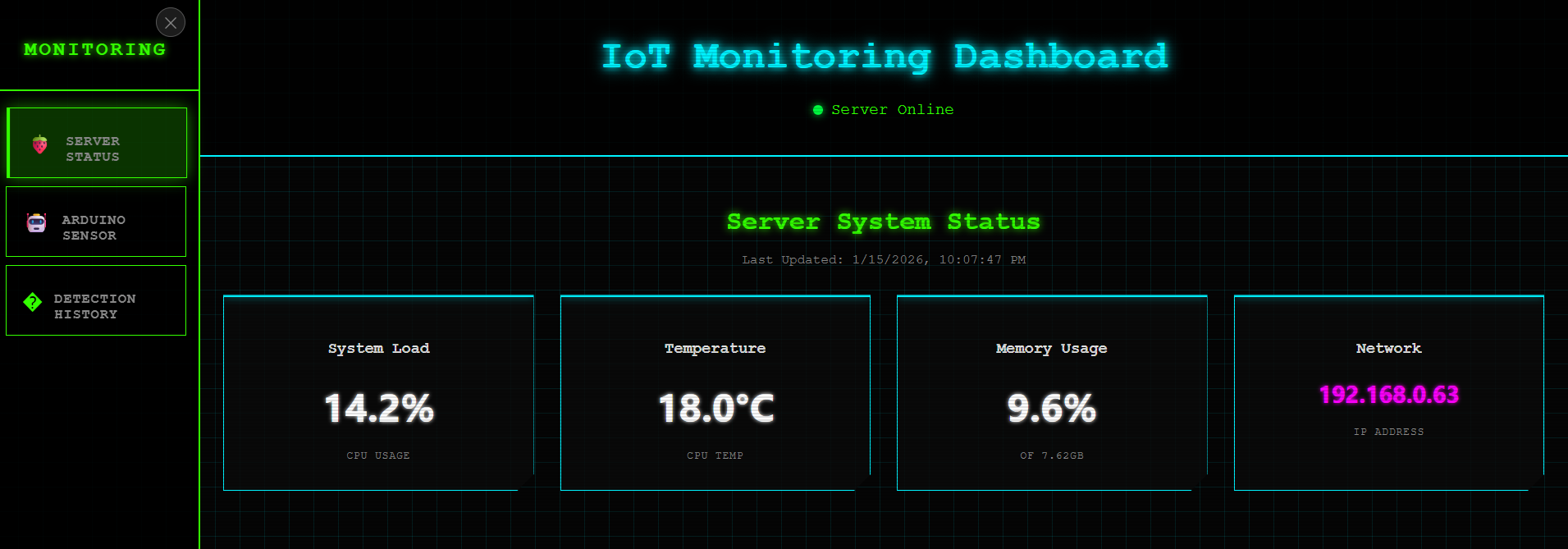
* + Real-Time Events: new detections via WebSocket with live updates
  + Video Playback: In-browser MP4streaming
  + Historical Data: Filterable detection log with date range
  + System status: Arduino heartbeat (WiFi RSSI, IP), Pi server stats (CPU, RAM, temperature)

The dashboard connects to the Pi server via Tailscale VPN for remote access.

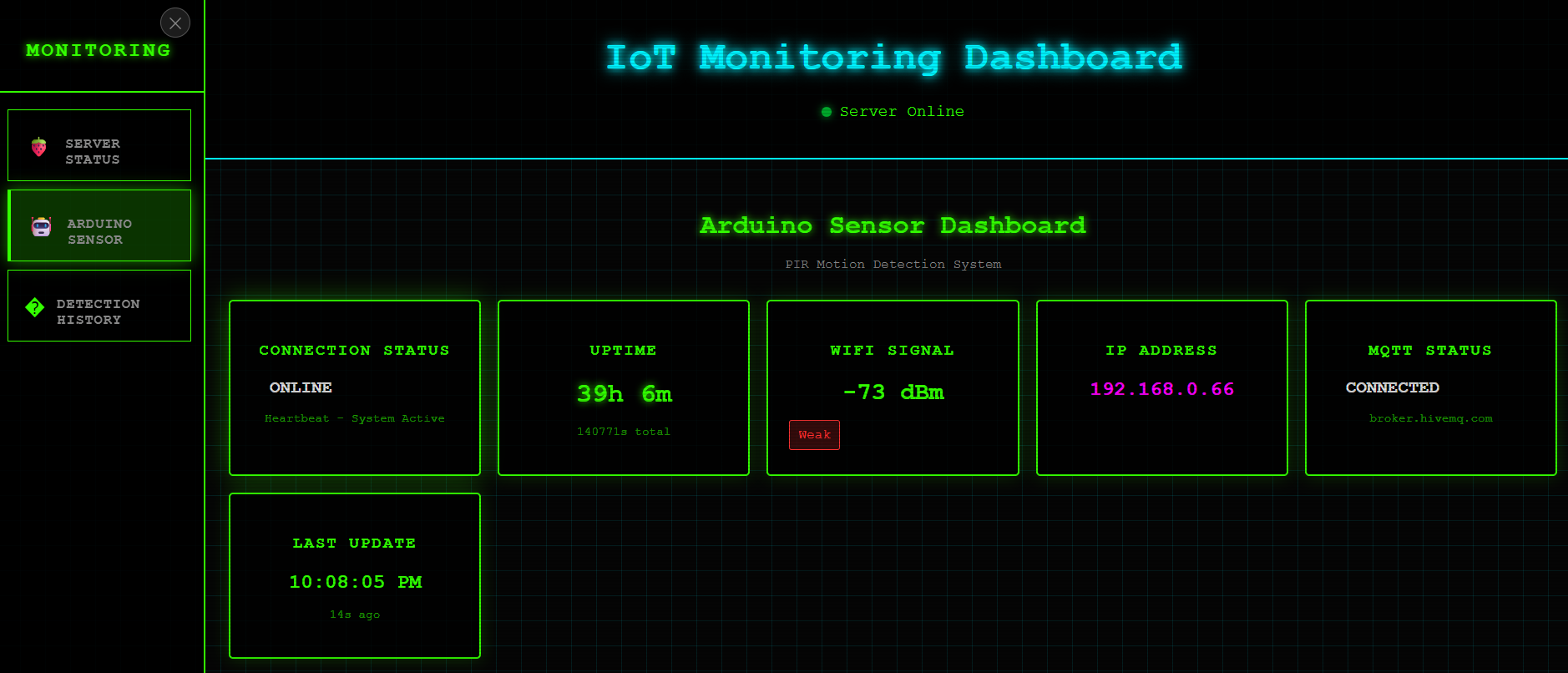
Live Updates:



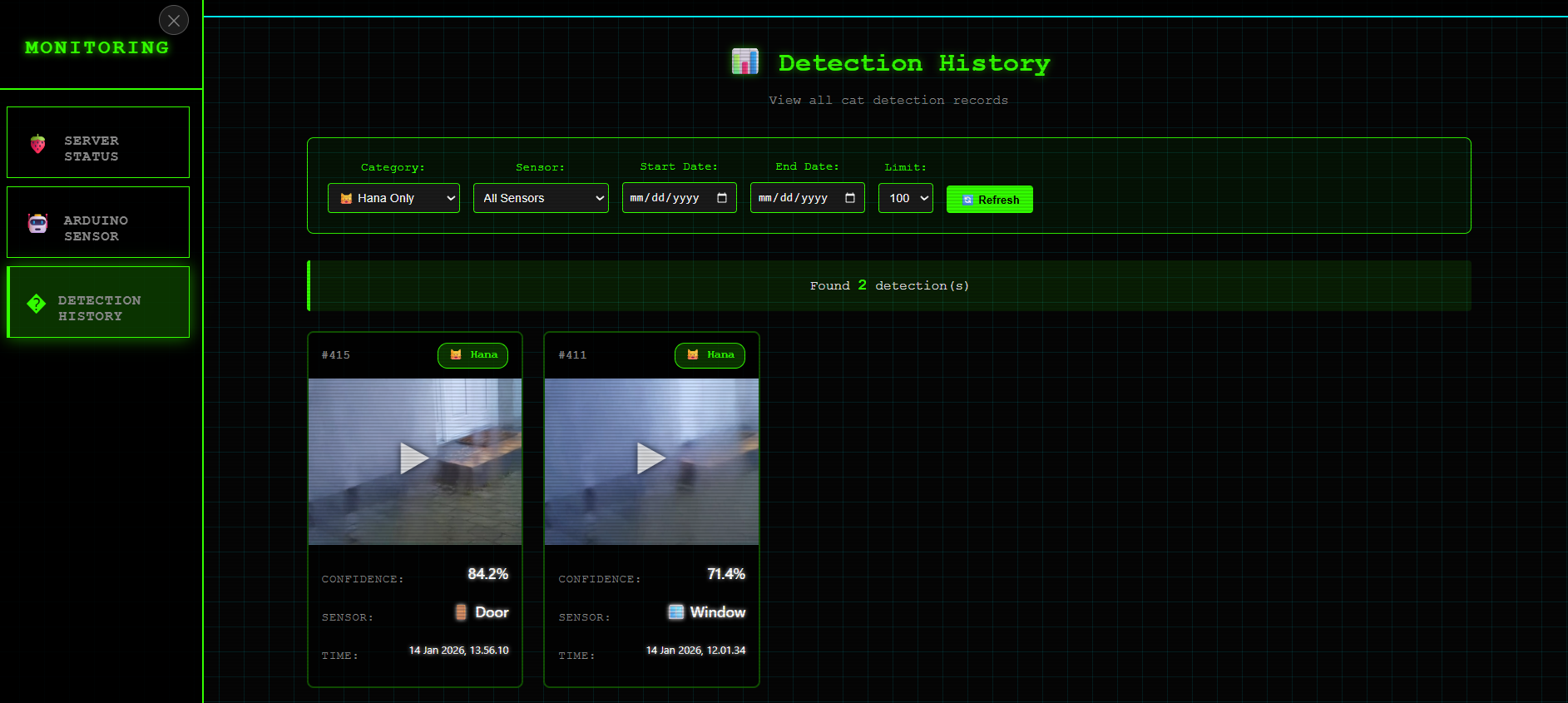
Server Monitoring:



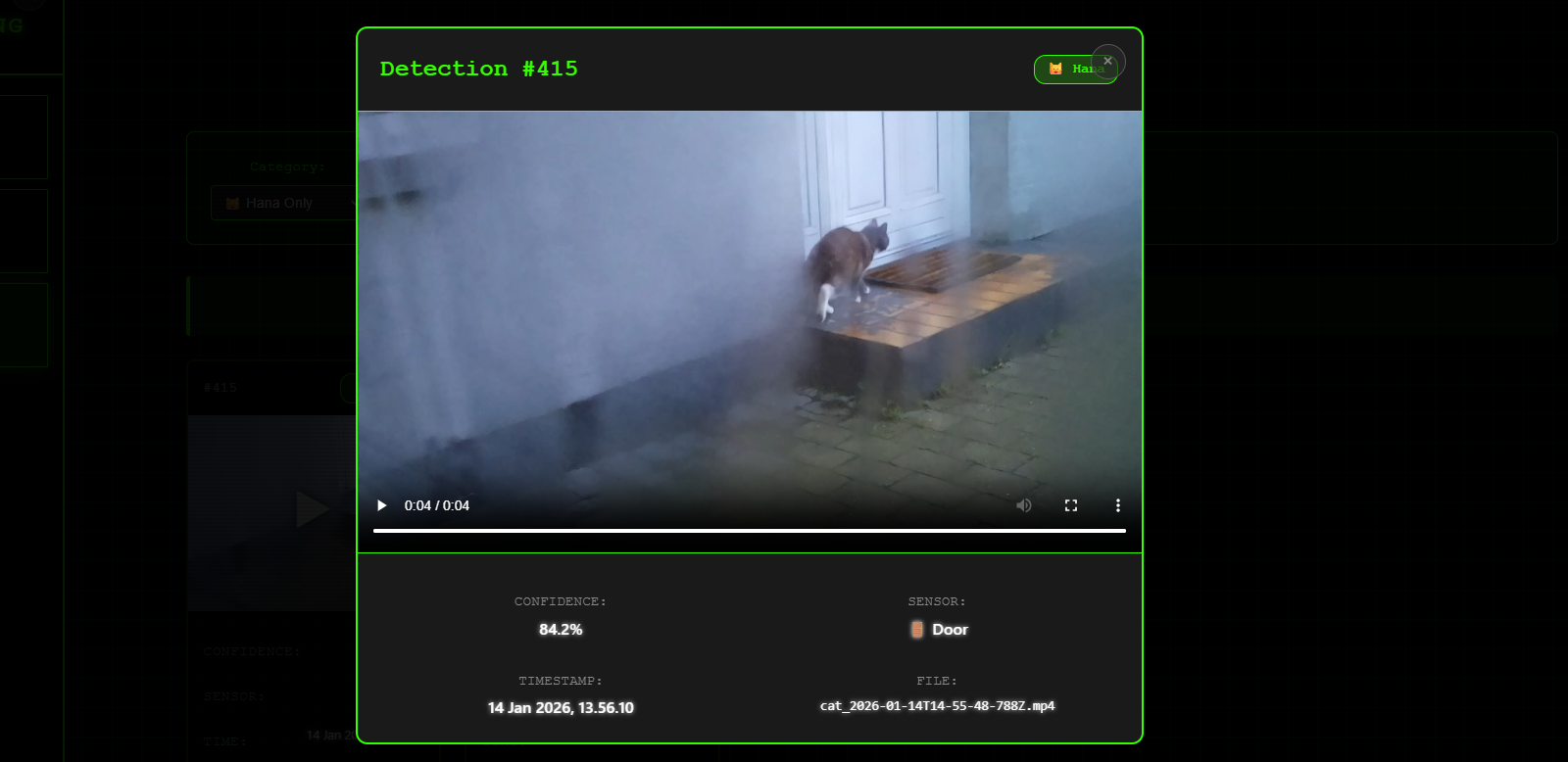
Arduino Monitoring:



Historical page with filter:



Video Playback with confidence score:



1. **Data Collection and Model training**

**6.1 Dataset:**

The model was trained on **151 images** of Hana (various poses, lighting conditions, angles) and **92 images** of other cats or with no cats for negative class.

Total: **243 training images**

**6.2 Training Approach:**

**Pattern-based Recognition**: Because the high camera angle often captures Hana from above, facial features are frequently not visible or are distorted. Therefore, the model was trained to focus primarily on **coat pattern recognition** that learning the distinctive color patterns, markings, and texture of Hana's fur rather than relying on facial features. This makes the detection more robust for overhead camera angles.

**Transfer Learning on Consumer Hardware**: Training was performed on a standard laptop computer using **transfer learning**, a technique where a pre-existing model (already trained on millions of images) is adapted to recognize Hana specifically. This approach reduces both the training time and the computational resources required compared to training a model from scratch. It also allows achieving good results with a relatively small dataset.

**Model Optimization for Raspberry Pi**: After training, the model was converted to TensorFlow Lite format (cat\_detector\_v1.tflite), which compresses it to approximately 4MB in size. This lightweight format enables fast inference on the Raspberry Pi's limited hardware, making real-time detection feasible without requiring expensive computing equipment.

1. **Results and Performance**

**7.1 System reliability**

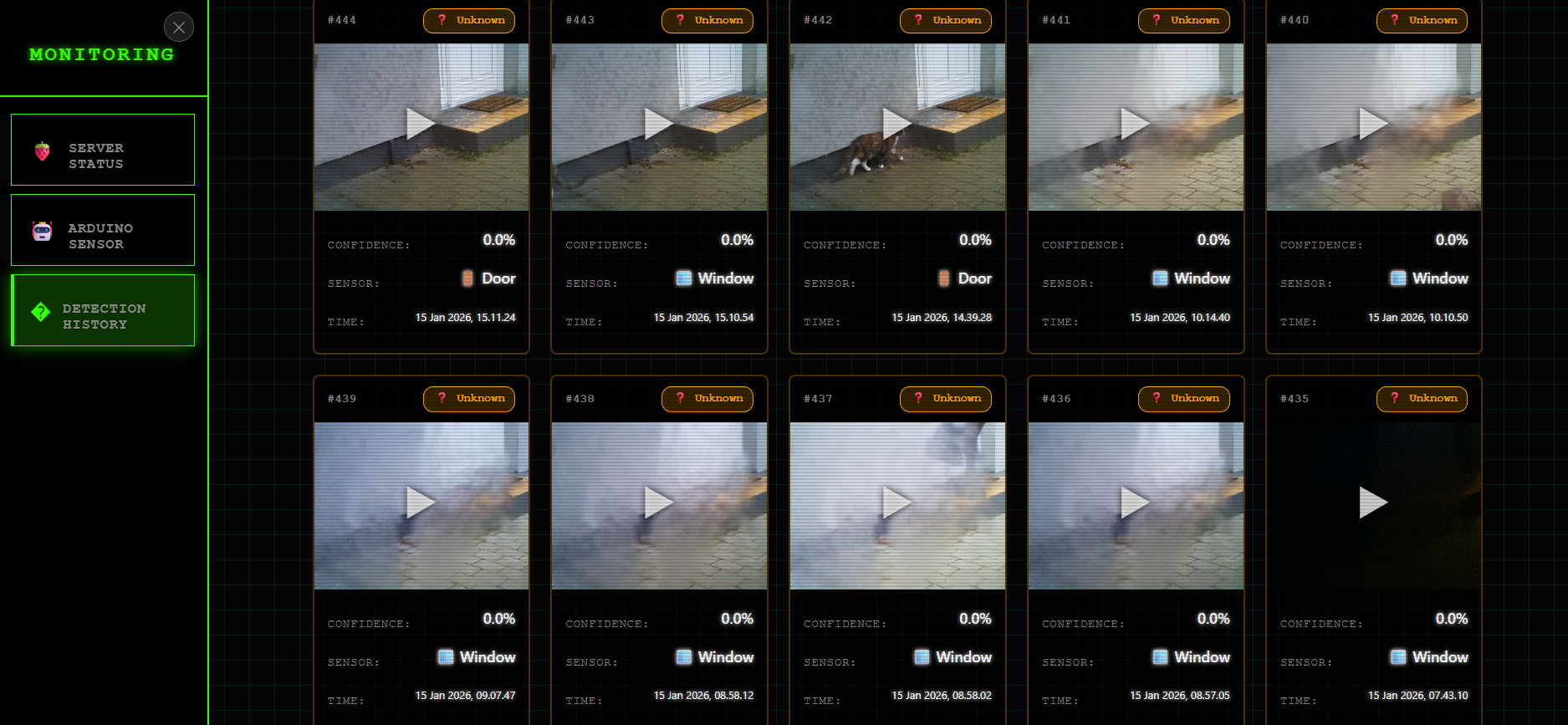
The PIR sensors reliably detect cat movement when approaching the door from the forward direction. However, cats coming from opposite direction (passing camera before sensor activation) results in empty videos classified as "no cat."

The camera locking mechanism prevents simultaneous recordings by allowing subsequent triggers within 5 seconds to share the current video capture.

**7.2 Classification Performance**

The model processes approximately 10 frames per video with 1.2 second total inference time. A 50% confidence threshold balances precision and recall.

However, the deployed system failed to reliably recognize Hana in real-world conditions. Classification accuracy faces several limitations: low-light conditions produce unreliable results due to camera hardware constraints, rain or snow on window glass significantly degrades image quality, and high camera angles may miss distinguishing coat patterns critical for identification. These deployment challenges highlight the gap between controlled training conditions and real-world outdoor operation.



1. **Conclusion**

This project demonstrates an end-to-end event-driven cat monitoring system combining edge devices (Arduino), edge computing (Raspberry Pi ML inference), and cloud connectivity (MQTT broker).  While the hardware and software infrastructure functions as designed, the ML classification model struggles to reliably identify Hana in real-world deployment, with most cat videos incorrectly classified as unknown.

Key achievements include a functional motion-triggered video capture system, real-time ML classification running on Raspberry Pi, a remote monitoring dashboard with historical data access, GitHub webhook auto-deployment that pulls code and restarts services when code from specific folder is modified and pushed from laptop, and robust error handling with automatic reconnection capabilities.

Future improvements could focus on following aspects.

Hardware upgrades would include a Pi Camera with IR capability for night vision, 3D-printed weatherproof housing for the PIR sensors with better placement to avoid empty videos, battery or solar panel power supply for the Arduino to eliminate the power cable and make mounting of device more convenient.

Software enhancements could expand the training dataset to 500+ images per class, implement object tracking to distinguish between "visit" and "stay" behaviors, add a notification system with email or push alerts for Hana detections, optimize video storage with automatic cleanup of old recordings, and configure a public domain for dashboard access instead of relying on private VPN,  and containerize services using Docker for easier deployment and portability.