Biodiversity Project Documentation

Technical documentation version 1

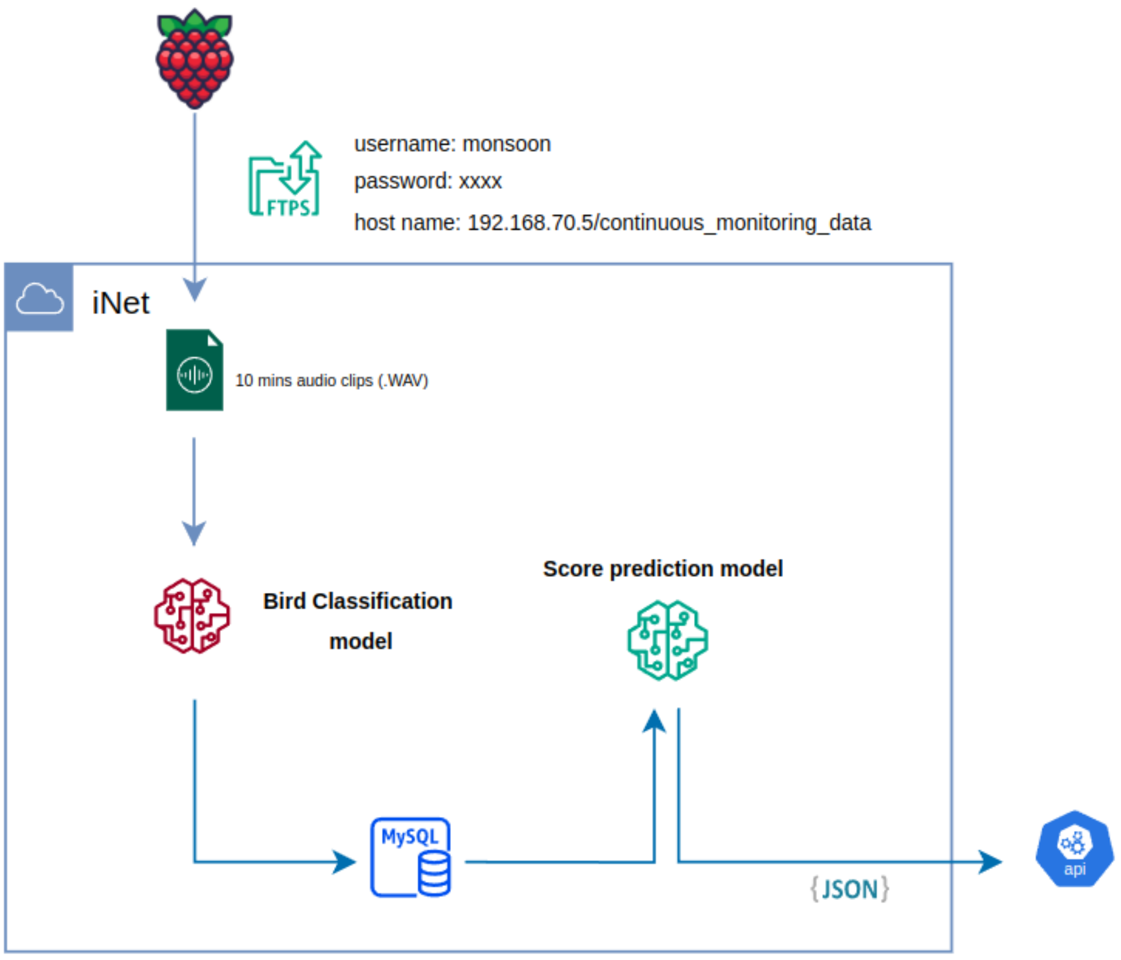
AIT

Table of contents

1. Biodiversity Project Documentation

Welcome to the documentation hub for the Biodiversity Project.

This covers end-to-end guidance for deploying IoT devices, training AI models for eco-acoustic analysis, and running the production inference pipeline for biodiversity monitoring and scoring. It includes a minimal setup path to get started quickly.


FTPS
username: monsoon
password: xxxx
host name: 192.168.70.5/continuous_monitoring_data
iNet
10 mins audio clips (.WAV)
Bird Classification model
Score prediction model
MySQL
{JSON}
api

High-level architecture of data collection, AI inference, and reporting.

1.0.1 What you'll find here

- Hardware setup and field deployment procedures
- AI model overview, datasets, and training approach
- Inference pipeline, database schema, and operations

2. Hardware Components & Setup Documentation

2.1 Device list

This system consists of the following IoT hardware components:

1. **Raspberry Pi 3 B+** (64 GB microSD storage)
2. **AudioMoth** (Acoustic logger)
3. **4G Router** (with SIM card)
4. **Internet SIM Card**
5. **Solar Panel** (with battery storage)

2.2 Raspberry Pi setup

2.2.1 Hardware

- **Model:** Raspberry Pi 3 B+
- **Storage:** 64 GB microSD card

2.2.2 OS installation

1. Insert the SD card into your computer.
2. Use **Balena Etcher** (or similar) to flash the provided `.img` backup OS image.
3. Insert the flashed SD card into the Raspberry Pi.
4. Power on the device.

Download Raspberry Pi OS Image

2.2.3 Login credentials

Default credentials (can be customized):
- **Username:** `pi`
- **Password:** `raspberry`

2.3 Example VPN configuration

Each Raspberry Pi has its own OpenVPN account.

2.3.1 File structure

Navigate to the OpenVPN directory:

```
cd /etc/openvpn/
ls
```

Expected files:

```
credentials.txt
openvpn_MONSOON_TEA05.conf
```

- **`openvpn_MONSOON_TEA05.conf`** → Converted `.ovpn` client config file
- **`credentials.txt`** → VPN username & password (two lines only)

2.3.2 Credentials setup

Check credentials:

```
cat /etc/openvpn/credentials.txt
```

Format:

```
vpn_username
vpn_password
```

Secure file permissions:

```
sudo chmod 600 /etc/openvpn/credentials.txt
```

2.3.3 Example VPN service setup

Enable & start service:

```
sudo systemctl enable openvpn@openvpn_MONSOON_TEA05
sudo systemctl start openvpn@openvpn_MONSOON_TEA05
```

Manual connect:

```
sudo openvpn --config openvpn_MONSOON_TEA05.ovpn
```

Verify connection:

```
ifconfig
```

VPN tunnel should point to `10.81.234.5`.

2.4 Configuration File Explanation (`config.json`)

This file defines the settings for the **bird sound monitoring device**. It covers FTP transfer, sensor recording behavior, system paths, and device identity.

Example **`config.json`**:

```
{
    "ftp": {
        "uname": "monsoon",
        "pword": "p8z3%1P#04",
        "host": "192.168.70.5/production-workflow-ec2",
        "use_ftps": 1
    },
    "offline_mode": 0,
    "sensor": {
        "sensor_index": 2,
        "sensor_type": "USBSoundcardMic",
        "record_length": 600,
        "compress_data": false,
        "capture_delay": 0
    },
    "sys": {
        "working_dir": "/home/pi/tmp_dir",
        "upload_dir": "/home/pi/continuous_monitoring_data",
        "reboot_time": "02:00"
    },
    "device_id": "00000000f1c084c2"
}
```

2.4.1 FTP Settings (`ftp`)

Controls how audio files are transferred from the device to the server.

Key	Value Example	Description
`uname`	`"monsoon"`	FTP username for authentication.
`pword`	`"p8z3%1P#04"`	FTP password for authentication.
`host`	`"192.168.70.5/production-workflow-ec2"`	FTP server address and target directory for uploads.
`use_ftps`	`1`	Enables **FTPS** (1 = secure FTPS, 0 = plain FTP).

2.4.2 Offline Mode (`offline_mode`)

Determines whether the device operates **without uploading data**.

- `0` → Online mode (default) – data is uploaded to server.
- `1` → Offline mode – data is only stored locally.

2.4.3 Sensor Settings (`sensor`)

Defines how the sensor captures audio.

Key	Value Example	Description
`sensor_index`	`2`	Index/ID of the sensor used (useful if multiple microphones are connected).
`sensor_type`	`"USBSoundcardMic"`	Type of sensor (here: USB sound card microphone).
`record_length`	`600`	Recording length in seconds (600 = 10 minutes).
`compress_data`	`false`	Whether to compress recordings before saving/uploading.
`capture_delay`	`0`	Delay before starting capture (in seconds).

2.4.4 System Settings (`sys`)

Defines working directories and reboot schedule for the device.

Key	Value Example	Description
`working_dir`	`"/home/pi/tmp_dir"`	Temporary folder for intermediate files.
`upload_dir`	`"/home/pi/continuous_monitoring_data"`	Directory where recordings are stored before upload.
`reboot_time`	`"02:00"`	Daily scheduled reboot time (HH:MM, 24-hour format).

2.4.5 Device ID (`device_id`)

- Example: `"00000000f1c084c2"`
- Unique identifier for the Raspberry Pi device.
- Used to distinguish recordings when multiple devices upload data to the same server.

2.5 Automatic recording service

This shellscript is setup to run the recording script at startup and reboot. So, the device will start recording as soon as it is powered on. And it will also reboot at the scheduled time. The shellscript is located at `/home/pi/custom-pi-setup/recorder_startup_script.sh`.

systemd service (`/etc/systemd/system/shellscript.service`):

```
[Unit]
Description=My Shell Script

[Service]
ExecStart=/home/pi/custom-pi-setup/recorder_startup_script.sh

[Install]
WantedBy=multi-user.target
```

Check live service logs:

```
journalctl -u shellscript.service -f
```

2.6 Important commands

Command	Purpose
`arecord -l`	List available recording devices
`journalctl -u shellscript.service -f`	Live monitoring of recording service
`sudo systemctl restart shellscript.service`	Restart recording service

2.7 AudioMoth setup

2.7.1 Overview

AudioMoth is a low-cost, full-spectrum acoustic logger, based on the Gecko processor range from Silicon Labs.
It can record **audible and ultrasonic frequencies** at rates from **8,000 to 384,000 samples/sec**.
It is used in two modes: **mobile** and **station**.

2.7.2 Modes

Mobile Type

- Portable configuration for temporary deployments
- Ideal for short-term surveys
  Download Mobile AudioMoth Manual (PDF)

Station Type

- Fixed position setup for continuous monitoring
- Powered by solar & external battery
  Download IoT Station Setup Manual (PDF)

2.8 Router setup & troubleshooting

- **Type:** 4G Router with SIM
- **Purpose:** Internet connection for remote locations

Troubleshooting Checklist: 1. Check LED status indicators
2. Ensure SIM card is active
3. Restart router if connection drops

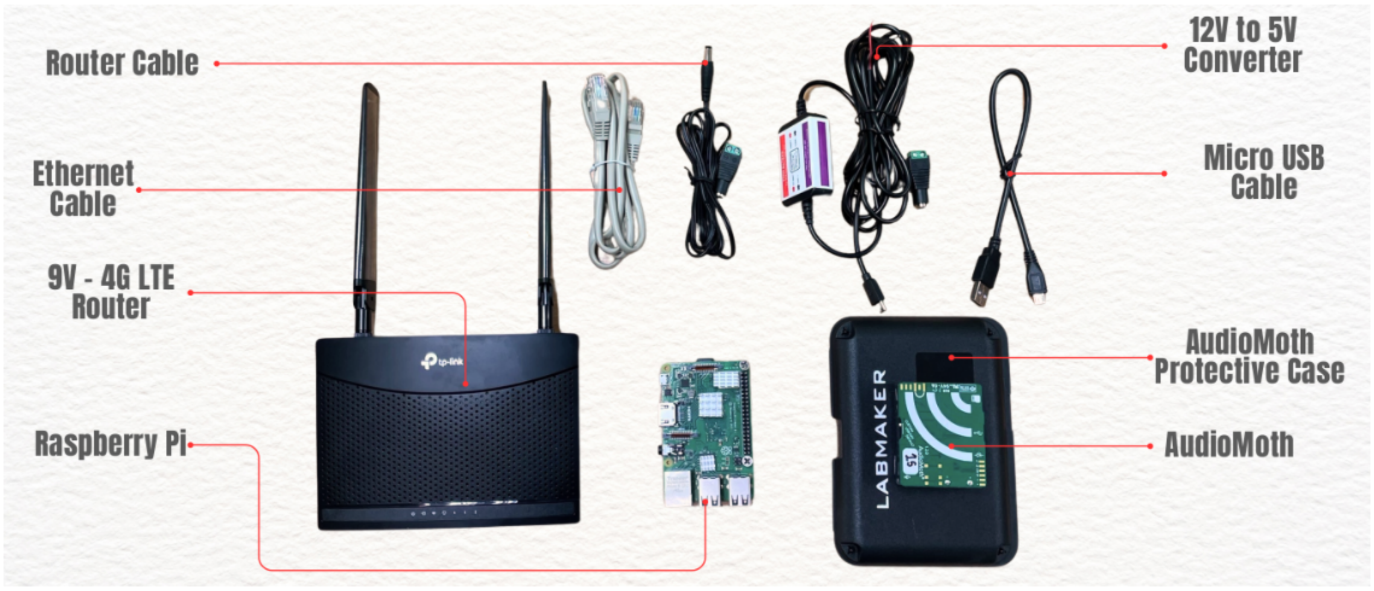
2.9 Solar panel & battery

2.9.1 Battery

- Stores energy for night/cloudy use
- **Blink Indicators:**
- 1 blink → Low
- 2 blinks → Medium
- 3 blinks → Full

2.10 System workflow

1. **Power Supply** → Solar Panel → Battery → Raspberry Pi & Router
2. **Data Capture** → AudioMoth or Raspberry Pi records audio
3. **Data Transmission** → Router sends via 4G
4. **Remote Access** → VPN connection for management
5. **Monitoring** → Logs checked via `journalctl` or SSH


Router Cable
Ethernet Cable
9V - 4G LTE Router
Raspberry Pi
12V to 5V Converter
Micro USB Cable
AudioMoth Protective Case
AudioMoth

3. Biodiversity Score Prediction

3.1 Outlines

3.2 Overview of Biodiversity Score Level Prediction

This project predicts regional biodiversity scores through a two-stage workflow Figure 1: **1. bird and insect sound classification** and the **2. biodiversity score level prediction**.

1. Bird and Insect Sound Classification Audio recordings are collected by deployed AudioMoth devices. As illustrated in Figure 1, the recordings are preprocessed and fed into a deep learning classifier based on a modified implementation of the BirdCLEF 2023 4th Place Solution. The model identifies bird and insect species and also detects non-biological sounds such as human speech, other human-generated noises, and vehicle sounds. It was pre-trained on species recordings from Xeno-canto and noise/no-call recordings from our own data collection.
2. Biodiversity Score Level Prediction For each region, the frequency of occurrence of every detected species and noise class is aggregated from the classified recordings. These frequencies serve as input features to a traditional machine learning model (XGBoost), which predicts the region's biodiversity score level: high, medium, or low.

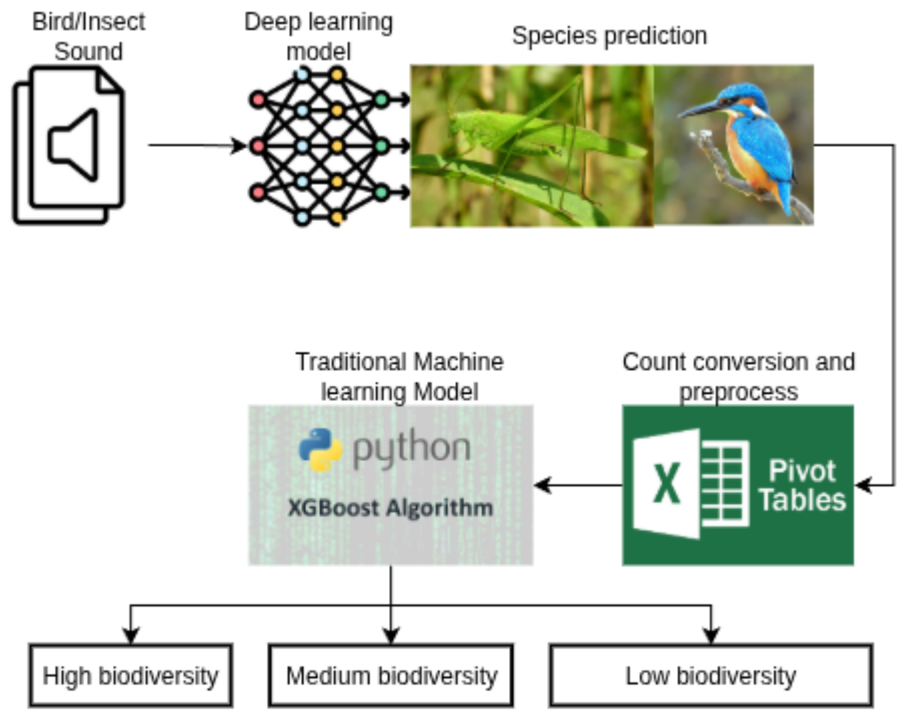


Figure 1: Biodiversity score level prediction overview.

3.3 Data

This section provides an overview of the data used in this project. We summarize the sources, label quality, use in training, geographic filtering, and the class list.

3.3.1 Sources

Public: Expert-labeled wildlife audio from Xeno-canto, covering birds and insects. Self-collected: Field recordings captured with AudioMoth devices in tea plantations around Chiang Mai, Thailand.

3.3.2 Audio Data Labels

Xeno-canto recordings include expert-provided species labels. The self-collected recordings lack ground-truth annotations.

3.3.3 Data Usage

The sound-classification model is trained primarily on the labeled Xeno-canto data. Additional noise examples from our self-collected recordings (e.g., human speech, noises from human activity, vehicles, and other environmental noises) are included to improve robustness.

3.3.4 Geographic filtering

To reduce label noise and improve relevance, we removed species not known to occur in Thailand, with a particular focus on the Chiang Mai region.

3.3.5 Class list

The final set of bird and insect species used in training and inference is documented in **species.txt** and **species_count.txt**.
The number of classes included in the analysis is summarized in Table 1.

3.3.6 Table 1 — Summary of the Number of Classes in Training Data

Type of Sound	Class Count
Bird Species	66
Insect Species	14
Noise / No-Call	10
Total	**90**

3.4 Models

3.4.1 Sound Classification Model

3.4.2 Biodiversity Score Level Prediction Model

- **Input:** Mel-spectrograms extracted from audio recordings.
- **Feature Extraction:** Convolutional Neural Networks (CNNs) for spatial feature learning.
- **Classification Layer:** Fully-connected layers with softmax output for multi-class classification.
- **Training Details:**
- Optimizer: Adam
- Loss: Categorical Cross-Entropy
- Learning Rate: *(e.g., 0.001)*
- Epochs: *(e.g., 50)*
- Batch Size: *(e.g., 32)*

4. Bird Sound Monitoring & Scoring Pipeline

4.1 Overview

This system automates the end-to-end process of **monitoring bird sounds** using IoT devices, classifying them with a soundscape model, predicting biodiversity scores, and delivering the results as JSON payloads to an API.

Repository: **inference-workflow-iNet**

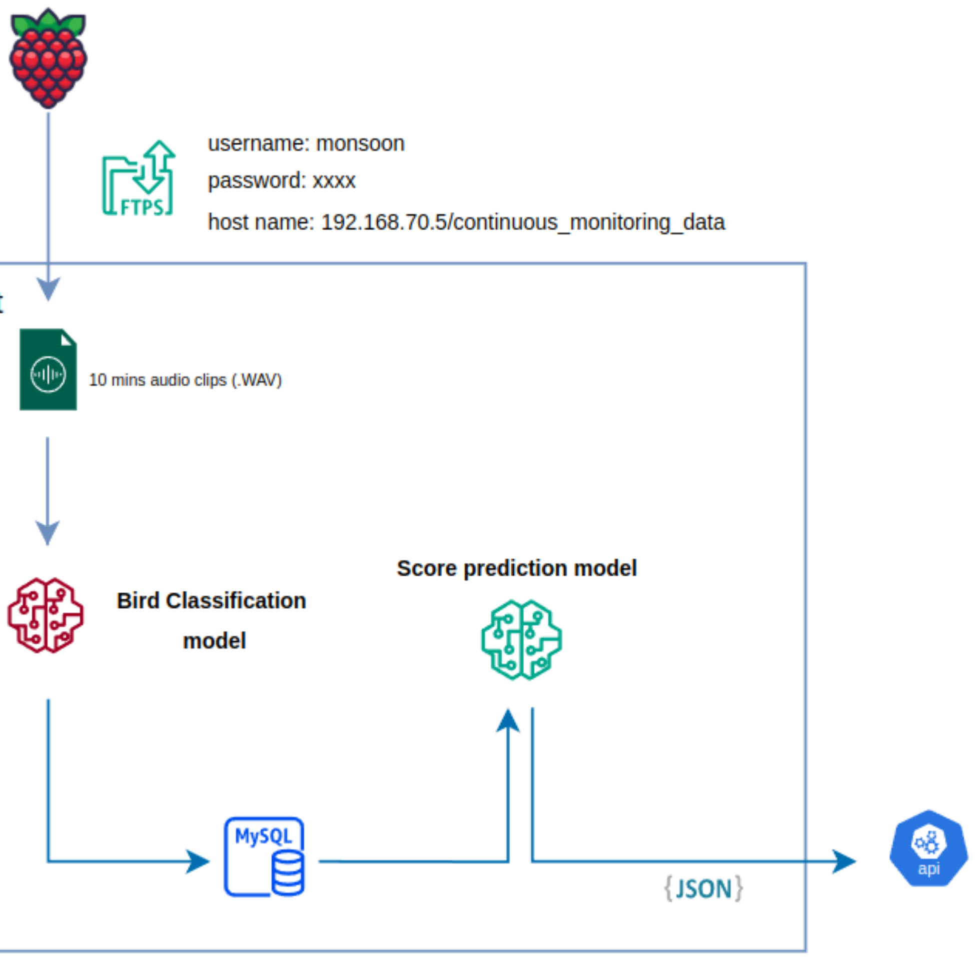

FTPS
username: monsoon
password: xxxx
host name: 192.168.70.5/continuous_monitoring_data
iNet
10 mins audio clips (.WAV)
Bird Classification model
Score prediction model
MySQL
{JSON}
api

Figure 1: Bird Sound Monitoring & Scoring Pipeline Overview

4.1.1 Data source – Raspberry Pi

- **Device:** Raspberry Pi with Audiomoth sensors for continuous field audio collection.
- **Protocol:** FTPS (Secure FTP) for encrypted data transfer.

- **iNet Server 4 Machine Information:**

Parameter	Value
Username	monsoon
IP Address	192.168.70.5
Password	p8z3%1P#04

- **Destination:** Audio files are securely uploaded to the **iNet private cloud**.

4.1.2 Audio collection

- Captures **10-minute audio clips** in `.WAV` format.
- Sample audio files can be downloaded from **this Google Drive link**
- See attached how to fetch audio from filezilla setup manual **Filezilla setup manual**

4.1.3 Bird classification model

- Processes audio clips using a **deep learning soundscape model**.
- Identifies bird species with **confidence scores**.
- Stores classification results in a **SQLite database** for further analysis.

4.1.4 Score prediction model

- Retrieves classification results from SQLite.
- Generates a **biodiversity score** for the monitored area based on detected species:
- **Score A** – High biodiversity
- **Score B** – Medium biodiversity
- **Score C** – Low biodiversity
- Outputs results to an API in structured JSON format.

Component	Description
Raspberry Pi + Audiomoth + 4G Router	Collects audio data from the field
FTPS	Secure file transfer protocol for uploading audio files
iNet Server	Runs inference processes for species detection
Bird Classification Model	Analyzes audio clips and identifies bird species + confidence
SQLite	Stores classification results and associated metadata
Score Prediction Model	Generates biodiversity scores from classification results
API	Receives JSON payloads containing final scoring results

4.2 Pipeline file structure

```
inference-pipeline/
├─── app-data/                  # Database files
├─── audio-data/                # Input audio recordings
├─── docker/                    # Docker configuration files
├─── json-output/               # Prediction results and reports
├─── logs/                      # Application logs
├─── monsoon_biodiversity_common/ # Core library modules
│    ├─── config.py              # Model and system configuration
│    ├─── dataset.py             # Data loading and preprocessing
│    ├─── db_model.py            # Database models and schema
│    ├─── model.py               # Neural network architecture
├─── scripts/                   # Utility and deployment scripts
├─── src/                       # Main application source code
│    ├─── audio_monitoring.py    # Real-time audio monitoring
│    ├─── process_detections.py  # Detection processing and reporting
│    └─── species_mapping.py     # Species classification mapping (Thai - Eng)
├─── weights/                   # Pre-trained model weights
└─── requirements.txt           # Main project libraries
```

4.3 Components

4.3.1 Main functions

- **File Monitoring**: Watchdog-based file system monitoring for automatic processing
- **Audio Processing**: Real-time audio file monitoring and processing
- **Deep Learning Models**: Attention-based neural network for species classification and xgboost model for prediction the score of the area.
- **Database**: SQLite database for storing detections results
- **Docker Support**: Containerized deployment for development and production

Note: This deployment is already set up on iNet and running in production. Use this guide to deploy elsewhere.

File Monitoring

The monitoring service is implemented in **`src/audio_monitoring.py`**.
Its function is to detect new audio files, check stability, and run inference.

- **Watched path:** `/app/audio-data/` (including subfolders with `RPiID-*`)
- **Supported formats:** `.wav`, `.ogg`, `.mp3`
- **Queueing:** Uses `watchdog` with a thread-safe `queue` for sequential processing

-**File stability check**
The system performs a file stability check before processing any new audio file. It waits until the file size is both non-zero and remains unchanged for at least three seconds. If the file does not stabilize within a maximum wait time of 60 seconds, it is considered unstable or empty and is skipped.

Inference process
The inference process uses the model stored at `/app/weights/soundscape-model.pt`. Each audio file is resampled to **48 kHz** and then divided into **5-second segments**. For every segment, the system predicts the class with the highest probability and records its confidence score. Each prediction is assigned a unique segment ID in the format `<file_stem>_<second>`.

Logging and retention
All activity is logged in `/app/logs/audio_inference.log`. After processing, the audio files are retained by default, although a safe delete option exists but is disabled.

Audio processing & detection model

This stage takes the daily detections from the database, turns them into structured features, predicts biodiversity scores, and delivers results to the API.

- For aggregation and scoring, detections are read from the SQLite database each day and grouped by device and hour. An XGBoost model (`/app/weights/xgboost-model.pkl`) processes these counts, fills in any missing species with zeros, and predicts a score. The device's daily score is then determined by the majority of its hourly predictions.
- For species information, each detected species is categorized as either a **bird** or an **insect**, with both English and Thai names provided from `species_mapping.py` .
- The final results are saved as JSON files in `json-output/predictions_<YYYY-MM-DD>.json` . Each file contains the **device ID, date, location, daily score, and detailed per-species data**.
- These results are also sent to the API using **OAuth authentication**. If delivery fails, the system retries up to **10 times** with exponential backoff. All activities related to result delivery are logged in `/app/logs/daily_report.log` .
- Example Payload per Device

```
{
  "<DEVICE_ID>": [
    {
      "date": "YYYYMMDD",
      "coordinate": [18.8018, 98.9948],
      "score": "A",
      "species": [
        {
          "name_en": "Blue-throated Barbet",
          "name_th": "นกโพระดกคางฟ้า",
          "type": "bird",
          "data": ["0", "1", "45", "23", "56", "12", "2", "0", "5", "9", "19", "21", "22", "34", "61", "23", "12", "6", "87", "112", "22", "46", "23", "11"]
        }
      ]
    }
  ]
}
```

Deep learning models

This AI models can be read from AI Models.

Database schema

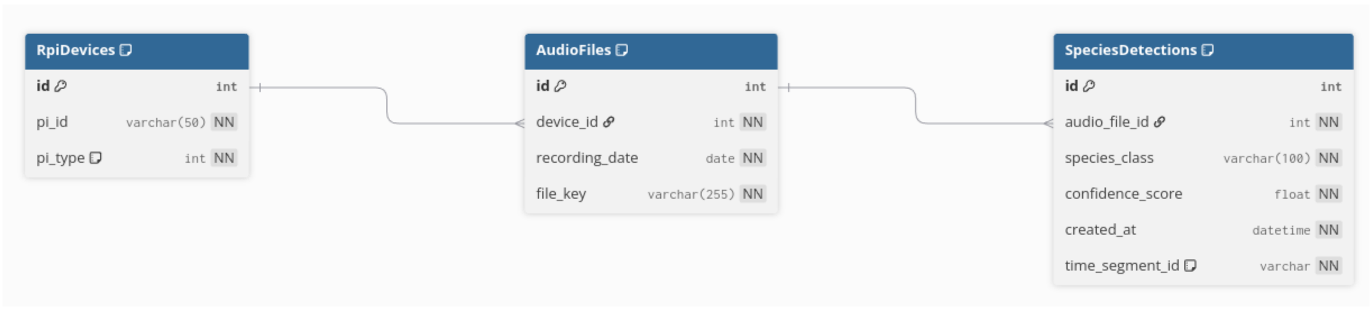
The database schema is defined using SQLAlchemy ORM. The schema consists of three main tables, each with relationships and constraints as described below:

1. RpiDevices - **Purpose:** Stores information about each Raspberry Pi device. - **Key Columns:** - `id` (Primary Key) - `pi_id` (Unique string identifier for the device) - `pi_type` (Integer: 0 = Mobile, 1 = Station; default is 1) - **Relationships:** One-to-many with `AudioFiles` (a device can have multiple audio files).

2. AudioFiles - **Purpose:** Records metadata for each audio file collected. - **Key Columns:** - `id` (Primary Key) - `device_id` (Foreign Key to `RpiDevices.id`) - `recording_date` (Date of recording) - `file_key` (Unique file identifier) - **Constraints:** Unique constraint on (`device_id` , `recording_date` , `file_key`) to prevent duplicate entries per device and date. - **Relationships:** One-to-many with `SpeciesDetections` (an audio file can have multiple detections).

3. SpeciesDetections - **Purpose:** Stores the results of species detection for each audio file segment. - **Key Columns:** - `id` (Primary Key) - `audio_file_id` (Foreign Key to `AudioFiles.id`) - `species_class` (Detected species name) - `confidence_score` (Detection confidence, float) - `created_at` (Timestamp of detection) - `time_segment_id` (String identifier for the segment within the audio file) - **Relationships:** Many-to-one with `AudioFiles` .

Entity Relationship Overview: - Each `RpiDevices` entry can have multiple `AudioFiles` . - Each `AudioFiles` entry can have multiple `SpeciesDetections` .



Docker support

1. **Build production image**

```
sh scripts/build_prod_image.sh
```

2. **Run with docker-compose**

```
docker compose -f docker/docker-compose-prod.yaml up -d
```

3. **Access container**

```
docker exec -it prod-bio-service bash
```

4. **Check logs**

```
docker logs -f prod-bio-service
```

4.4 Example usage

4.4.1 Installation

1. **Clone the repository (with submodules)**

```
git clone <repository-url>
cd inference-pipeline
git submodule update --init --recursive
```

2. **Download model weights** Download from Google Drive and place in the `weights/` directory:
3. `weights/soundscape-model.pt` — sound classification model
4. `weights/xgboost-model.pkl` — score prediction model
5. Google Drive: Weights link
6. Ensure the model architecture matches `monsoon_biodiversity_common/config.py`
7. **Verify logs (JSON sending and daily detections)**
8. Follow container logs:

```
docker logs -f prod-bio-service
```

9. Inspect detailed log files inside the container:

```
docker exec -it prod-bio-service bash
tail -n 200 -f /app/logs/audio_inference.log   # file monitoring & segment inference
tail -n 200 -f /app/logs/daily_report.log      # daily aggregation & API sending
```

10. Expected entries:
    - Saved JSON: lines with "[SAVE] JSON saved to"
    - Successful API send: lines with "[API] Prediction sent"
    - Daily aggregation: lines with "[DATE] Generating report for:" and device/file counts
    - Real-time processing: lines with "[NEW FILE]", "[INFER] Processing", and "[DB] Added detections"