Biodiversity Project Documentation

Techinal documentation version 1

AIT

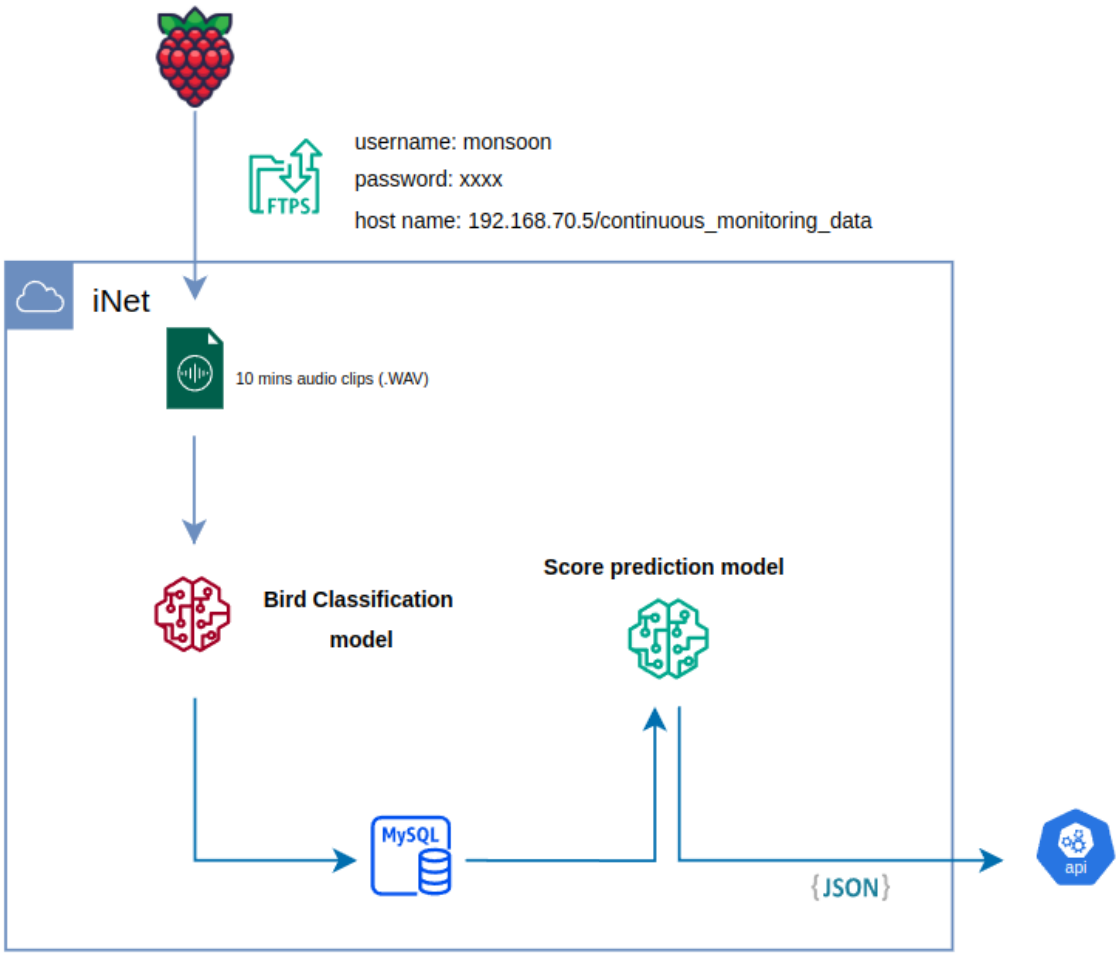
None

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.



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 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:

```
client/
credentials.txt
openvpn_MONSOON_TEA05.conf
server/
update-resolv-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 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 Device configuration file

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"
}
```

2.5 Automatic recording service

Example **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 Solar panel

- Powers IoT devices in remote areas
- **Indicators:**
- Green → Charging
- Red → Low battery
- Off → No power

2.9.2 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
LABMAKER

3. Biodiversity score prediction

3.1 Overview

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 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.

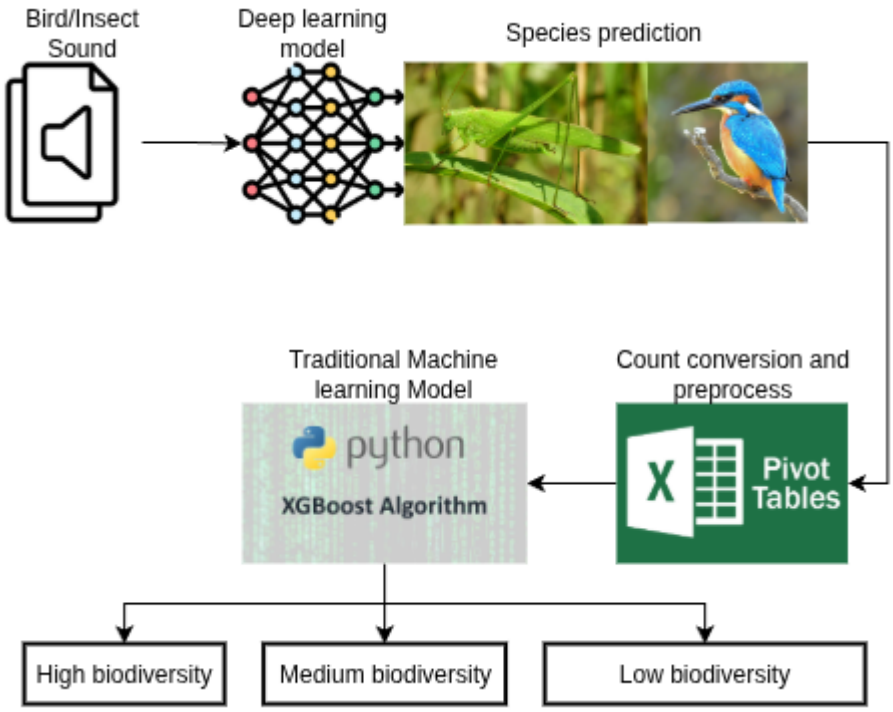


Figure 1: Biodiversity score level prediction overview.

3.2 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 final class list.

3.2.1 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.

3.2.2 Label quality

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

3.2.3 Use in training

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, human activity, vehicles, and other environmental noises) are included to improve robustness.

3.2.4 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.

3.2.5 Class list

The final set of bird and insect species used in training and inference is documented in species.txt.

3.2.6 Training dataset

The dataset consists of **bird species**, **insect species**, and **noise classes**.

Bird species

Common Name	Biological Name	Number of Samples
Asian Koel	Abroscopus-superciliaris	118
(Add more rows here)		

Insect species

Common Name	Biological Name	Number of Samples
Cicada	Cicadidae	800
(Add more rows here)		

Noise classes

Class Name	Description	Number of Samples
(Example) Rain	Background rain noise	500
(Add more rows here)		

3.3 Model architecture

The sound classification system is based on:

- **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**

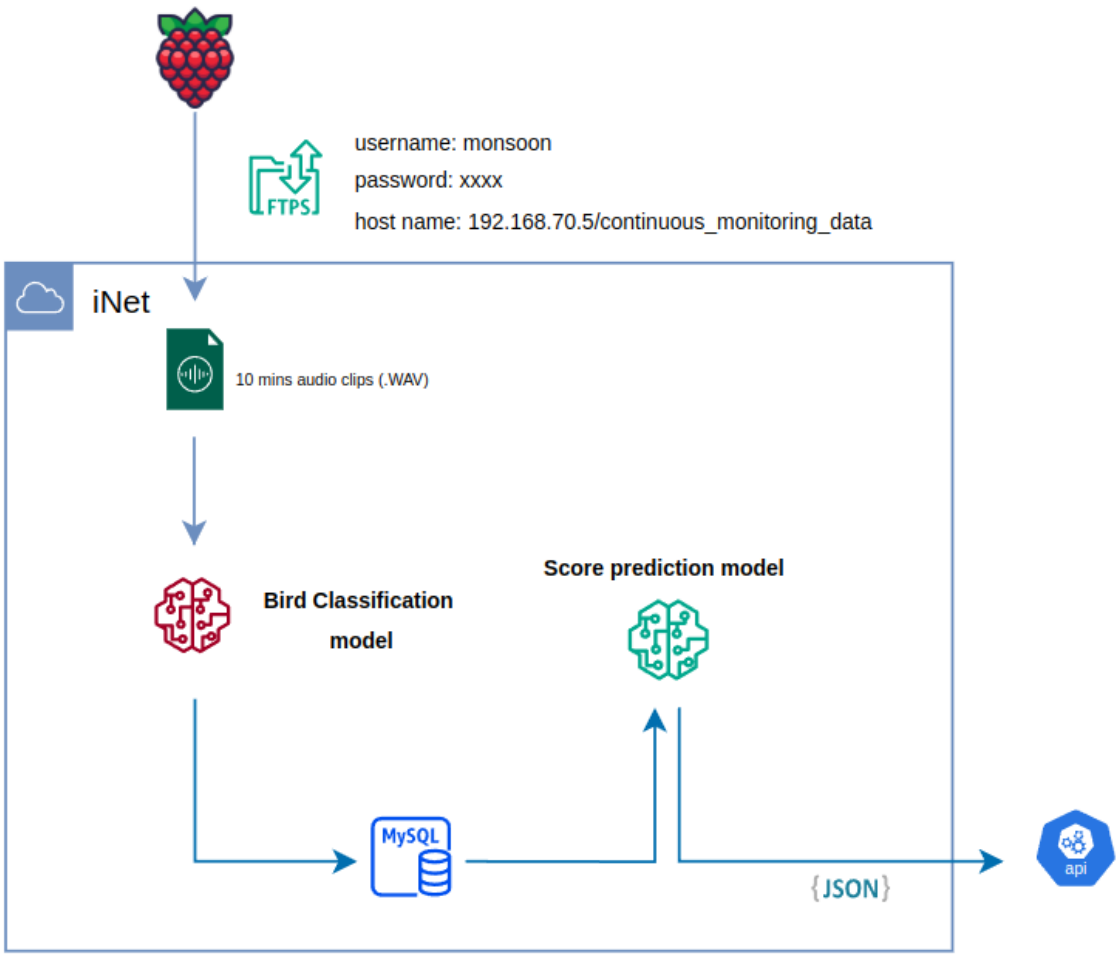


Figure 1: Bird Sound Monitoring & Scoring Pipeline Overview

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

Location:
`src/audio_monitoring.py`

Purpose:
Continuously watches incoming audio, validates file stability, performs 5-second window classification, and writes per-segment detections to the database with resilient logging.

Watched paths and filters: - **Root directory:** `/app/audio-data/` - **Subfolders monitored:** Only folders whose names contain `RPiID` (e.g., `RPiID-001`), monitored recursively - **File types:** `.wav`, `.ogg`, `.mp3`

Monitoring and queueing: - Uses `watchdog` (`Observer` + `FileSystemEventHandler`) to enqueue new files into a thread-safe `queue.Queue` - A dedicated background thread consumes the queue and processes files sequentially

File stability gate: - Before inference, the service waits for the file size to remain unchanged and non-zero for 3 consecutive seconds - Maximum wait window: 60 seconds; unstable or empty files are skipped with an error log

Inference details: - Loads `AttModel` with weights at `/app/weights/soundscape-model.pt` - Audio loaded with `librosa` at 32 kHz; split into 5-second segments: seconds = 5, 10, 15, ... up to clip length - For each segment, computes sigmoid probabilities for `CFG.target_columns` and selects: - `Class`: argmax label - `Score`: max probability - Segment identifier `row_id` format: `<file_stem>_<second>`

Database write (SQLite): - DB URL: `sqlite:////app/app-data/soundscape-model.db` - Entities used: `RpiDevices`, `AudioFiles`, `SpeciesDetections` - Path parsing: expects folder layout `.../RPiID-XXX/YYYY-MM-DD/<audio_file>` - Unique `AudioFiles.file_key`: `<pi_id>_<recording_date>_<audio_filename>` - For each segment, inserts a `SpeciesDetections` row with: - `time_segment_id` = `row_id` - `species_class` = predicted `Class` - `confidence_score` = `Score` - `created_at` = current UTC timestamp

Logging: - Console: Colored via `colorlog` - File: `/app/logs/audio_inference.log`

File retention: - Files are currently kept after processing; a safe delete helper exists but is disabled by default

Summary:
Real-time, resilient ingestion that only processes fully-written files, performs 5-second window classification, and persists detections per segment.

Audio processing & detection model

Locations:
`src/process_detections.py`, `src/species_mapping.py`

Daily aggregation and scoring: - Loads an XGBoost model from `/app/weights/xgboost-model.pkl` - Reads detections from SQLite (`sqlite:///app-data/soundscape-model.db`) for a target date (default: yesterday) - Builds a feature table per device and hour by counting detections per species - Ensures a fixed feature set (`SELECTED_FEATURES`), filling missing species with zeros - Predicts per-row scores and assigns the device score by majority vote across its hourly rows

Hourly bucketing logic: - `AudioFiles.file_key` encodes a start time: `..._<YYYY-MM-DD>_<HH>-<MM>-<SS>` - For each `SpeciesDetections.time_segment_id` (`..._<relative_second>`), computes `absolute_second = start_time_in_seconds + relative_second` - Buckets into `hour = clamp(floor(absolute_second / 3600), 0..23)`; fallback `hour=0` if parsing fails

Species categorization and localization: - Species are tagged as `bird` or `insect` via sets in `process_detections.py` - English/Thai display names are pulled from `SPECIES_INFO` in `species_mapping.py`

JSON output (saved and sent): - Saves to `json-output/predictions_<YYYY-MM-DD>.json` - Payload per device:


```
{
  "<DEVICE_ID>": [
    {
      "date": "YYYYMMDD",
      "coordinate": [18.8018, 98.9948],
      "score": 5,
      "species": [
        {
          "name_en": "Blue-throated Barbet",
          "name_th": "นกโพระดกคางฟ้า",
          "type": "bird",
          "data": ["0", "1", "0", "0", "0", "0", "0", "0", "0", "0", "0", "0", "0", "0", "0", "0", "0", "0", "0", "0", "0", "0", "0", "0", "0"]
        }
      ]
    }
  ]
}
```


API delivery and retries: - Obtains OAuth token using env vars: `TOKEN_URL`, `CLIENT_ID`, `CLIENT_SECRET`, `API_USERNAME`, `API_PASSWORD` - Sends JSON to `API_URL` with `Authorization: Bearer <token>` - Up to 10 attempts with exponential backoff; skips sending if token cannot be obtained - Logs to `/app/logs/daily_report.log`

CLI usage: - `python process_detections.py --date YYYY-MM-DD` — run for a specific date - `python process_detections.py --now` — run immediately for today - `python process_detections.py --schedule` — run daily at 23:59

Deep learning models

This AI models can be read from AI Models.

Database schema

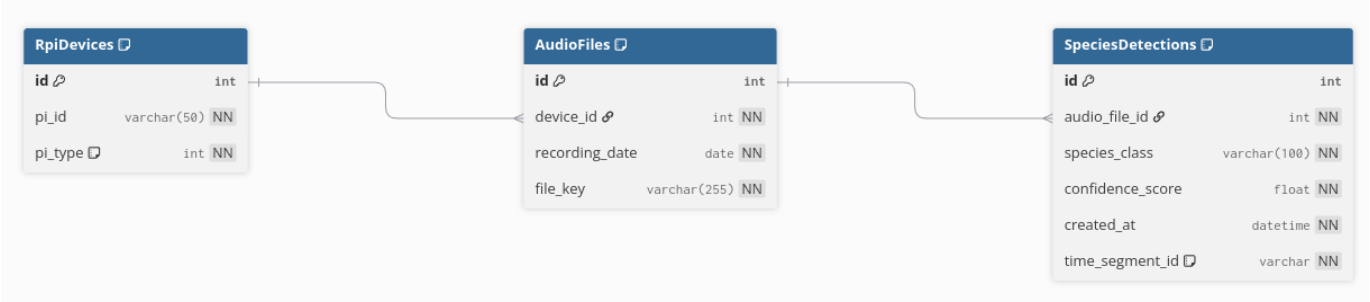
The database schema is defined using SQLAlchemy ORM. The schema consists of three main tables, each with relationships and constraints as described below:

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. **Build and run with Docker / Docker Compose**
8. Build via script:

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

9. Or build with Docker directly:

```
docker build -t prod-bio-service -f docker/Dockerfile .
```

10. Start with Docker Compose:

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

11. Check containers:

```
docker ps
docker compose -f docker/docker-compose-prod.yaml ps
```

12. **Verify logs (JSON sending and daily detections)**
13. Follow container logs:

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

14. 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
```

15. 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"