UCLouvain

École polytechnique de Louvain

EchoExplorer, A Web Application for Automated Bat and Wildlife Call Classification

Authors: **Anthony Gerlache, Noam Sirjacobs**
Supervisors: **Olivier Bonaventure, Maxime Piraux**
Reader: **Lucile Dierckx**
Academic year 2023–2024
Master [120] in Computer Science

Acknowledgments

First and foremost, we would like to thank our Master's thesis promotor, Prof. Olivier Bonaventure for his guidance and expertise.

We would like to extend our thanks to our very helpful assistant, Maxime Piraux for his advice on technical aspects as well as his support and time.

We would also like to thank Lucile Dierckx for her time and assistance with the technicalities of BatML and the creation of the ensemble model.

We are deeply appreciative of the feedback given by Jean Philippe Lefin and Julien Otoul.

A special thanks to Yves Laurent for providing us with a very helpful labelled dataset.

ChatGPT has been used in some parts of this document as a tool to help formulate and correct specific sections.

And lastly, to our friends and families, thank you for your support and encouragements.

Abstract

This master thesis presents the development of EchoExplorer, a web application aimed at facilitating automated bat call detection. Traditional methods and existing AI tools for classifying bat calls require technical expertise or advanced computer science skills, limiting their accessibility.
EchoExplorer serves as an invaluable tool for bat enthusiasts. By creating a web application, we allow users to get rid of technical complexities and have access to sophisticated AI-driven analysis.

Based on an existing AI model created during two previous master theses[7][17], the application offers and can support the integration of various AI models for different animal species, not limited to bats. The user interface provides multiple tools to give users a better experience while analysing their audio. Additionally, our application offers collective analysis by allowing users with expertise in bats identification to review and validate the results generated by the AI which enables labelled data collection, a key point in machine learning, often missing.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Bats are essential components of various ecosystems and play a crucial role in maintaining the balance of insect populations. However, many bat species are endangered, making it imperative to monitor their populations closely.
Traditional methods of monitoring bat activity are tedious and require substantial expertise. While there are software tools available for visualising and analysing audio data, and several AI models have been developed for classifying bat calls, these solutions often need some level of technical expertise to operate. This creates a barrier for many bat researchers, amateurs and conservationists who may not have advanced computer science skills.

In previous researches at Université catholique de Louvain, several thesis projects laid the groundwork for this thesis. A first thesis introduced Batmen [17], an AI system capable of detecting and classifying bat calls within audio files. Building on this, a subsequent thesis created BatML [7], an AI model capable of multi-label classification. Both BatML and Batmen were trained on datasets provided by Natagora, a leading conservation organisation, amongst other datasets.

The objective of this thesis is to bridge the gap between advanced AI tools and their practical application by non-experts in the field of bat research.
We developed EchoExplorer, a web application that integrates BatML as well as other AI models, enabling users to upload and process their audio files with minimal effort. Our platform not only facilitates the detection and classification of bat calls but also provides visualisation tools, in the form of waveform and spectrogram displays.

This allows experts users to validate the AI-generated labels, ensuring the reliability of the data collected by general users and creating new data that can be used in further research.

EchoExplorer employs a range of technologies to achieve its objectives. The frontend is built using JavaScript and HTML, we use the WaveSurfer.js library, providing an intuitive interface for audio visualisation. The backend is powered by Python's Flask library, with data storage managed through Amazon S3 buckets and MongoDB. Some main features of the application include the ability to annotate and label regions of audio files, request AI processing for call detection, and share annotated files within a community which experts can then validate.

EchoExplorer is designed to be modular, allowing for the addition of new AI models with minimal effort. The visualisation aspects are flexible enough to be relevant for birds, amphibians or insects if corresponding AIs were to be added to the application.

In conclusion, EchoExplorer represents a step towards making AI tools accessible to a wider audience. By enabling easy visualisation, labelling, and AI-assisted analysis of audio data, we provide a valuable resource for bat amateurs, conservationists and researchers, supporting their vital work in preserving these endangered species.

Chapter 2

State of the Art

This chapter presents other existing tools that either automatically perform bat call detection and identification, enable visualisation, which allows performing these tasks manually, or facilitate data gathering.

2.1 Audacity

Audacity[27] is an open-source digital audio editor and recording software, which is available on Windows, macOS and Linux. At first sight, Audacity might not be very useful for bat call detection and classification given that it is not its objective, however it is worth to note that Audacity can show an oscillogram as well as a spectrogram of the audio, both of which are very useful for manual bat call identification. An oscillogram is a visual representation of the signal's intensity over time, while a spectrogram is a visual representation of the signal's intensity over both time and frequency. Indeed, these are great tools for manual animal identification, since oscillograms can focus on more precise measurement of changes in amplitude, while spectrograms can highlight low and high frequency vocalisations. Although neither is superior to the other in terms of information provided, spectrograms is often the favourite tool to perform animal identification thanks to its frequency aspect.
Audacity also allows users to select parts of the audio and write some notes, which is a great feature for experts who can use it to perform manual identification.

Figure 2.1 shows what oscillogram and spectrogram look like on Audacity. Under these, there are examples of the notes that can be written with Audacity.

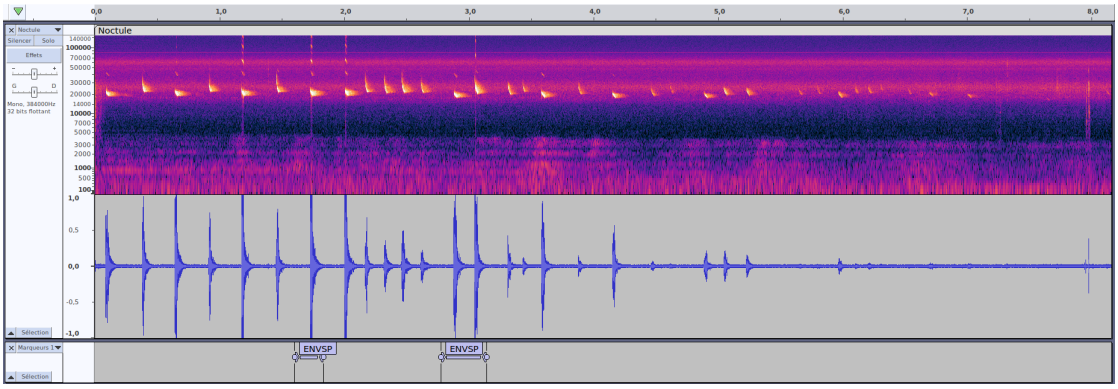


Figure 2.1: Screenshot of Audacity. The audio file shown is a recording of bats from the Nyctalus Noctula species.

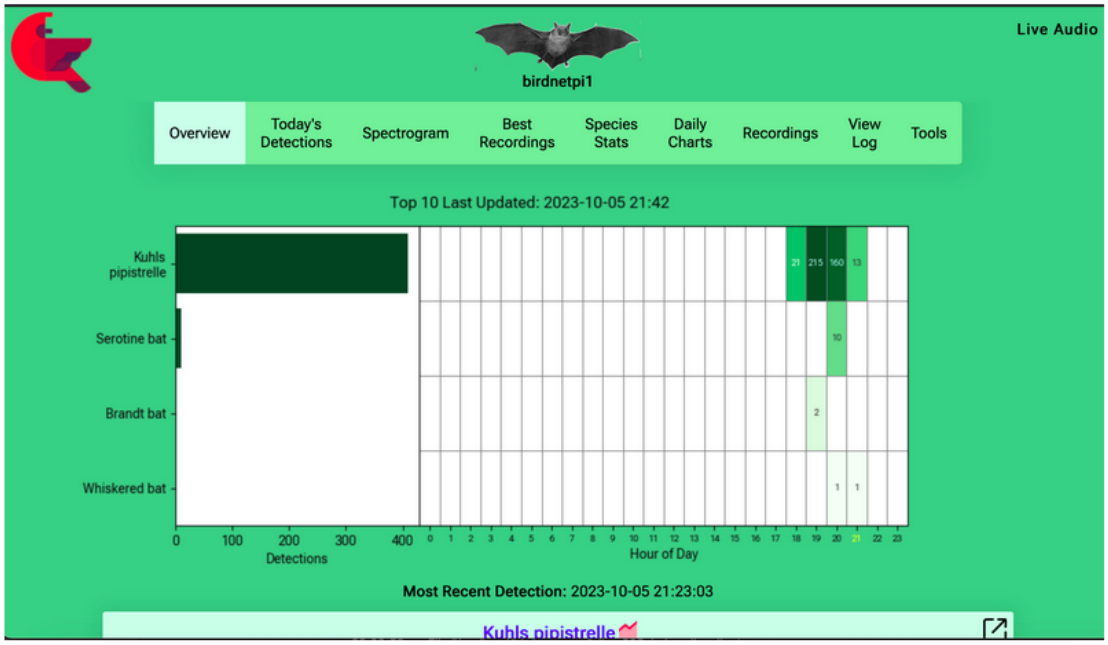
2.2 BattyBirdNET-Pi

BattyBirdNET-Pi[39] is an automated real time bat identificator. BattyBirdNET-Pi is a specialised version of the BattyBirdNET[38] application, designed to run on Raspberry Pi.

BattyBirdNET is a Python AI that performs bat call identification. It can be used on its own, but has no user interface, which can be a problem for people who are not comfortable with python. BattyBirdNET-Pi is built to run efficiently on Raspberry Pi, making it suitable for portable deployments and remote monitoring applications.
It allows users to record bat sounds in real time thanks to a microphone connected to the Raspberry Pi, and then perform the analysis of the recordings. The users can access a web interface where the results are displayed, such as the species detected, the number of detections and the time of each detection.

Unfortunately, BattyBirdNET-Pi is not so easy to use because it requires a Raspberry Pi, a recorder and some knowledge in computer science to be able to set up and install everything needed.

Figure 2.2 shows the main page of the web interface visualising the analysis of the recordings. The list of the detected species, as well as their detection times can be observed.



Source: BattyBirdNET-Pi Github [39].
Figure 2.2: Screenshot of BattyBirdNET-Pi.

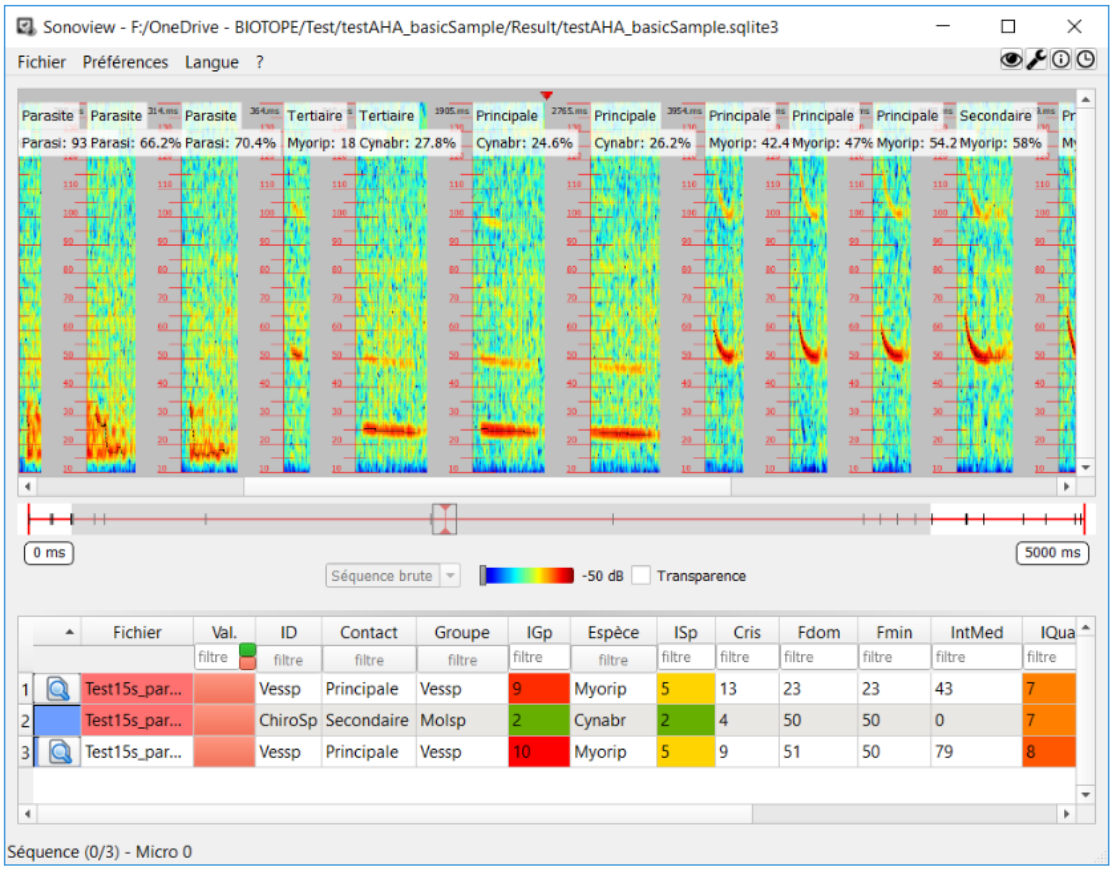
2.3 SonoChiro

SonoChiro[9] is a commercial software developed by Biotope[8], a research unit that has studied various areas. The primary motivation for creating this tool was to address the question of conservation for bats in different areas. This fundamental question led to several others, including which species are present, what is the state of their populations, what methods exist for counting them, what exactly is being counted, and what tools can be used for these counts. The team that created the software had a strong foundation in bat identification, thanks to the method developed by Michel Barataud [4].

SonoChiro performs both detection and classification of bat calls. After detecting the calls, the software outputs a species for every call found in the audio sequence. In addition to its performance, here is a non-exhaustive list of its advantages:

- It is compatible with every bat recorders on the market.
- It is able to process up to tens of Go of .WAV and .RAW files at the same time.

- It has different classifiers based on location.
- It supports mono and stereo files.
- It has a rich visualisation tool to view the results, with a spectrogram and a table listing all calls with many sorting options and important measures. Figure 2.3 illustrates these properties.



Source: SonoChiro [9].
Figure 2.3: Screenshot of SonoChiro.

Here are some of its drawbacks:

- It is a commercial software.
- It is only available on Windows.
- It is greedy in terms resources, minimum Intel I5 processor and 16 Go RAM are recommended.

To conclude, this software has a lot of great features, but because of its price, 1000€ for a license during one year, we were unable to test it and compare it with available AIs, nor do we have information about its classifiers and how they work.

2.4 Vigie-Chiro, Tadarida & ChiroSurf

Vigie Chiro [29] is a French monitoring program dedicated to the study and conservation of bats, coordinated by the Muséum National d'Histoire Naturelle.

In the context of this program, two software tools were developed: Tadarida [5] and ChiroSurf.

Tadarida is a toolbox for detection, labeling, and classification of acoustic recordings. It is made up of three modules, each built on top of the previous ones:

- The first module, **Tadarida-D**, handles detection.
- The second module, **Tadarida-L**, handles labeling.
- The last module, **Tadarida-C**, is used for classification.

BatDetect2, one of the AIs we use for EchoExplorer, is implemented using Tadarida-D for the detection process.

ChiroSurf is a software tool designed to visualise bat calls. It provides an interface for researchers and amateurs to inspect acoustic recordings and manually verify the automated identifications made by Tadarida.

2.5 BatClassify

BatClassify[12] is an open-source software available on Windows that performs bat call detection and classification. It has been trained on a subset of bat species from the UK. The classification uses a tree-based ensemble[20].

The software provides a minimal user interface that prompts the user to; (i) select the input folder containing files to analyse and (ii) select the output folder, in which the CSV file containing the results will be saved.

For each file, BatClassify outputs one line in the CSV file. This line indicates the presence probability of all species for which it has been trained.

Table 2.1, shows the list of species detected by BatClassify as well as their precision and recall scores obtained during the testing phase.

Species	Precision	Recall	N
Barbastella barbastellus	0.9955	0.9095	243
Myotis alcathoe	0.8300	0.8700	23
Myotis bechsteinii	1.0000	0.6250	16
Myotis brandtii/mystacinus	0.8611	0.9511	244
Myotis daubentonii	0.9944	0.8786	210
Myotis nattereri	0.9771	0.9624	133
NSL	0.9974	0.9872	391
Plecotus auritus	0.9831	0.8788	198
Pipistrellus pipistrellus	0.9901	0.9804	510
Pipistrellus pygmaeus	0.9644	0.9675	308
Rhinolophus ferrumequinum	1.0000	1.0000	79
Rhinolophus hipposideros	1.0000	1.0000	353

Source: BatClassify Bitbucket [12].
Table 2.1: Performance of BatClassify. NSL group comprises Nyctalus noctula, Nyctalus leisleri and Eptesicus serotinus.

Since BatClassify is open-source, we were able to test it on our own datasets presented in Chapter 5. Table 2.2 presents the results, and it can be noticed that the performances are quite worse than what was indicated in Table 2.1.

	Lauzelle	Xeno-canto	
Species	Recall	Precision	Recall
empty	0	1.0	0.71
Barbastella barbastellus	/	1.0	0.83
Myotis alcathoe	/	0	0
Myotis bechsteinii	/	0	0
Myotis brandtii/mystacinus	0.21	0.75	0.6
Myotis daubentonii	0.26	0.82	0.24
Myotis nattereri	0.50	1.0	0.67
NSL	0.93	0.75	0.75
Plecotus auritus	0.33	0.85	0.63
Pipistrellus pipistrellus	0.62	0.92	0.71
Pipistrellus pygmaeus	0.88	0.94	0.67
Rhinolophus ferrumequinum	/	0.71	0.63
Rhinolophus hipposideros	/	1.0	0.91

Table 2.2: Performance of BatClassify measured on our test datasets (see Section 5.2). NSL group comprises Nyctalus noctula, Nyctalus leisleri and Eptesicus serotinus.

2.6 Observations.org

Observations.org is a platform that allows users to record and share their nature observations, including birds, mammals, insects, plants, and fungi, among others.

The platform aims to gather biodiversity data through public participation, encouraging anyone to contribute their findings.
Users can submit observations with images, sounds, and location data, which can then be validated by experts to ensure data quality.

This validated data helps build a database of biodiversity, which can be used for scientific research, conservation efforts, and public awareness.

Figures 2.4 and 2.5 show the page dedicated to the Western Barbastelle.

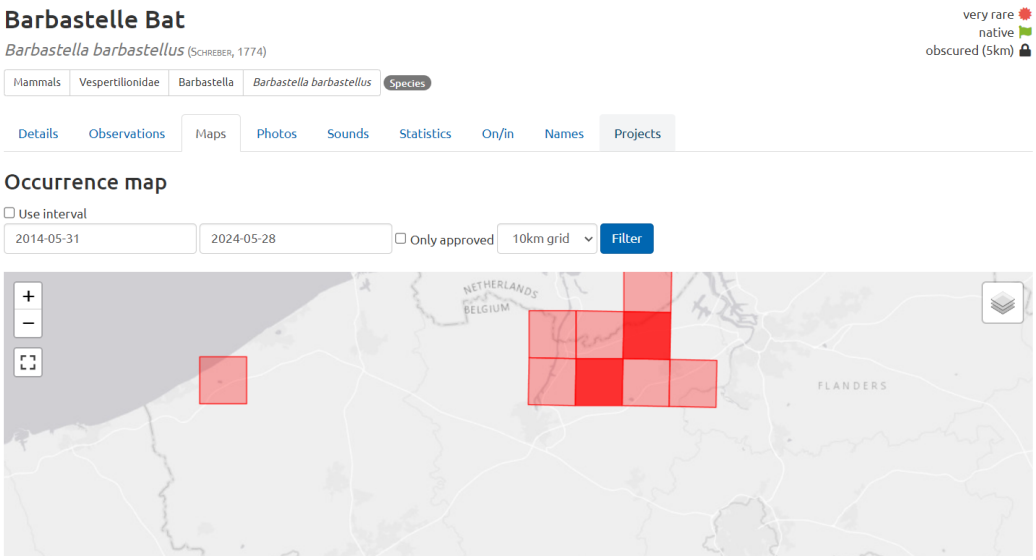


Figure 2.4: Observations.be, Map displayed on the Barbastelle bat page.

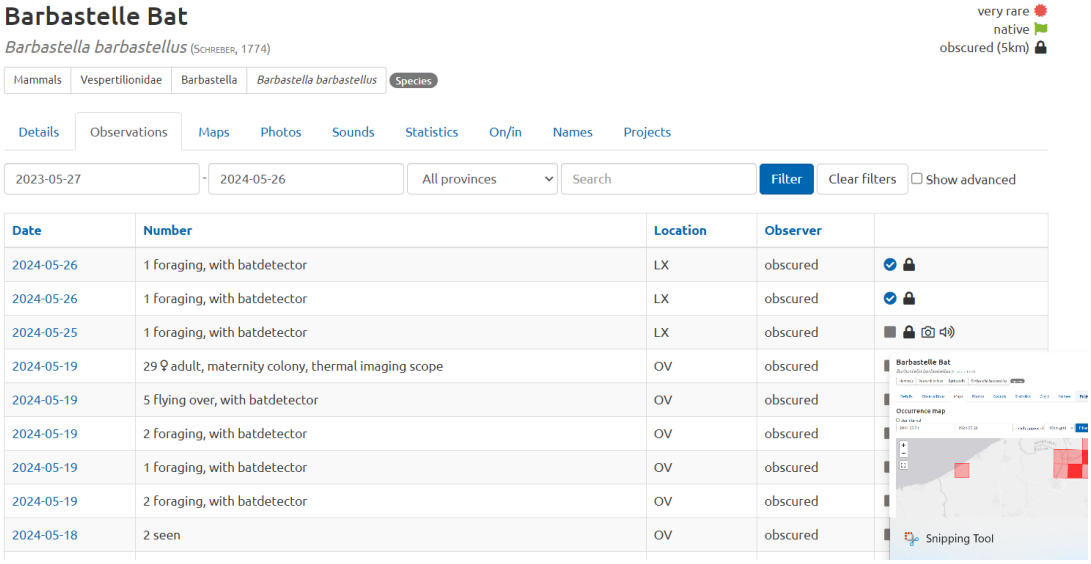


Figure 2.5: Observations.be, Table displayed on the Barbastelle bat page.

2.7 Xeno-Canto

Xeno-Canto [37] is another online platform and database dedicated to sharing bird, bat, frog and grasshopper sound recordings from around the world.

In a similar way it provides a platform for animal enthusiasts and researchers to upload, search, and download wildlife calls.

Xeno-Canto's extensive collection is freely accessible, making it a valuable resource for scientific research, conservation projects, and other educational purposes.

The platform displays a table with recordings and information on the type of call, locations, date, etc... As well as an intractable map showing the locations of the recordings. The sounds can be played from the website

Figures 2.6 and 2.7 show the page dedicated to the Western Barbastelle.

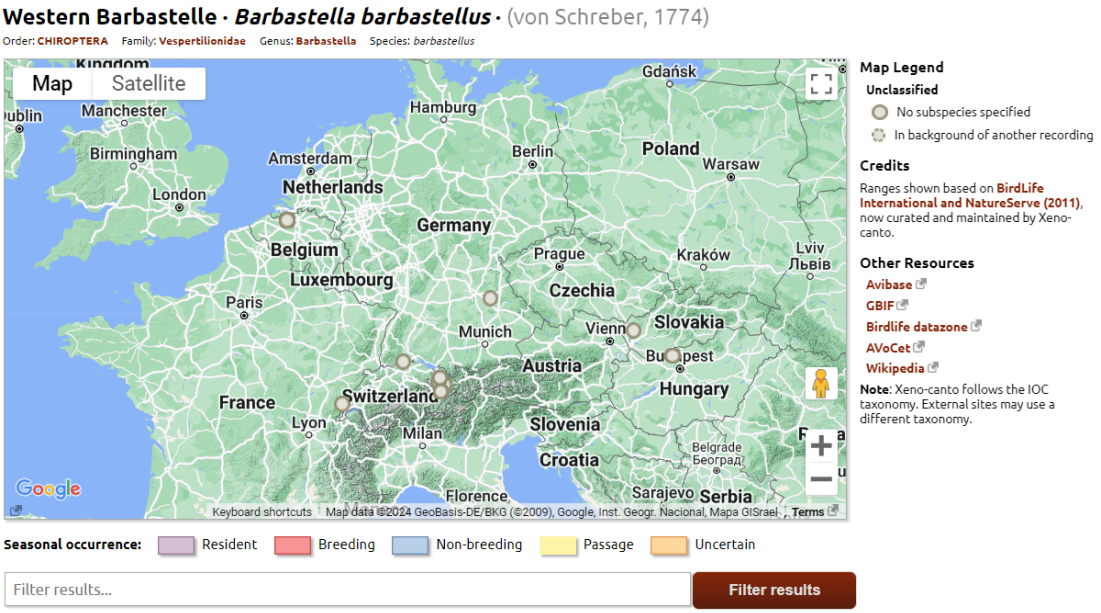


Figure 2.6: Xeno-Canto, Map displayed on the Western Barbastelle page.

1 2 Next ›

	Common name / Scientific	Length	Recordist	Date	Time	Country	Location	Elev. (m)	Type (predef. / other)	Remarks	Actions / Quality	Cat.nr.
▶ U	Western Barbastelle (*Barbastella barbastellus*)	0:11	Antonio Anta Brink	2023-09-30	20:30	Switzerland	Hallau, Unterklettgau, Schaffhausen	420	echolocation	open meadow in forest with pond nearby [sono]	A B C D E	XC833502
▶ U	Western Barbastelle (*Barbastella barbastellus*)	0:01	Vladimír Nemček	2023-09-15	21:30	Slovakia	Plavecké Podhradie, Malacky District, Bratislava Region	340	echolocation	mist-netting before the cave PP2, no… more » [sono]	A B C D E	XC830022
▶ U	Western Barbastelle (*Barbastella barbastellus*)	0:05	Niklas Palmcrantz	2023-08-05	00:00	Sweden	Gällerstena, Ydre Municipality, Östergötlands län	280	echolocation	[sono]	A B C D E	XC824684
▶ U	Western Barbastelle (*Barbastella barbastellus*)	0:13	Vladimír Nemček	2023-08-11	22:30	Slovakia	Kamenica nad Hronom, Nové Zámky District, Nitra Region	200	echolocation	mist-netting before the entrance of the… more » [sono]	A B C D E	XC822586
▶ U	Western Barbastelle (*Barbastella barbastellus*)	0:07	Vladimír Nemček	2023-08-11	22:30	Slovakia	Kamenica nad Hronom, Nové Zámky District, Nitra Region	200	echolocation	mist-netting before the entrance of the… more » [sono]	A B C D E	XC822585
▶ U	Western Barbastelle (*Barbastella barbastellus*)	0:04	Vladimír Nemček	2023-08-11	22:30	Slovakia	Kamenica nad Hronom, Nové Zámky District, Nitra Region	200	echolocation	mist-netting before the entrance of the… more » [sono]	A B C D E	XC822583

Figure 2.7: Xeno-Canto, Table displayed on the Western Barbastelle page.

2.8 WAV file format

Positions	Sample Value	Description
1-4	“RIFF”	Marks the file as a riff file. Characters are each 1 byte long.
5-8	File size (integer)	Size of the overall file - 8 bytes, in bytes (32-bit integer). Typically, you’d fill this in after creation.
9-12	“WAVE”	File Type Header. For our purposes, it always equals “WAVE”.
13-16	“fmt”	Format chunk marker. Includes trailing null
17-20	16	Length of format data as listed above
21-22	1	Type of format (1 is PCM) - 2 byte integer
23-24	2	Number of Channels - 2 byte integer
25-28	44100	Sample Rate - 32 byte integer. Common values are 44100 (CD), 48000 (DAT). Sample Rate = Number of Samples per second, or Hertz.
29-32	176400	(Sample Rate * BitsPerSample * Channels) / 8.
33-34	4	(BitsPerSample * Channels) / 8.1 - 8 bit mono2 - 8 bit stereo/16 bit mono4 - 16 bit stereo
35-36	16	Bits per sample
37-40	“data”	“data” chunk header. Marks the beginning of the data section.
41-44	File size (data)	Size of the data section.

Table 2.3: WAV (RIFF) File Header Structure.

2.9 Conclusion

This section has shown that there are several useful tools for bat call identification. However, many of these tools require computer science skills, such as proficiency in Python for BattyBirdNET or the ability to use a Raspberry Pi for BattyBirdNET-PI. Other tools, like SonoChiro, appear to be great but are not open-source, making them inaccessible for most bat enthusiasts. On the other hand, BatClassify is very easy to set up and use, though it has limitations in terms of performance.

We have also identified that audio visualisation tools for manual detection are available, along with platforms designed to gather, share, and validate data on wildlife sounds.

However, none of the tools listed above offer all three functionalities together, highlighting the need for EchoExplorer.

Chapter 3

Web application

3.1 Global structure

Our web application is structured as a client-server architecture with a HTML/-JavaScript front-end for user interaction and a Python back-end to deal with file processing and database interactions.

For our front-end we mainly used four important libraries: (i) `Bootstrap5.js` [30] to have more visually uniform elements across the application and a way to arrange them; (ii) `WaveSurfer.js` [23] to help represent the waveforms and spectrograms of the audios; (iii) `Leaflet.js` [1] to use interactive maps; (iv) `DataTables.js` [14] to help us filter and search through our results.

The back-end was developed with Flask [33], a well known Python web framework, and the server is hosted by an Apache server. All these libraries were chosen for their simplicity and efficiency of use, and more details will come in the following sections.

Figure 3.1 illustrates the global structure of the web application, with each part referenced to its section.

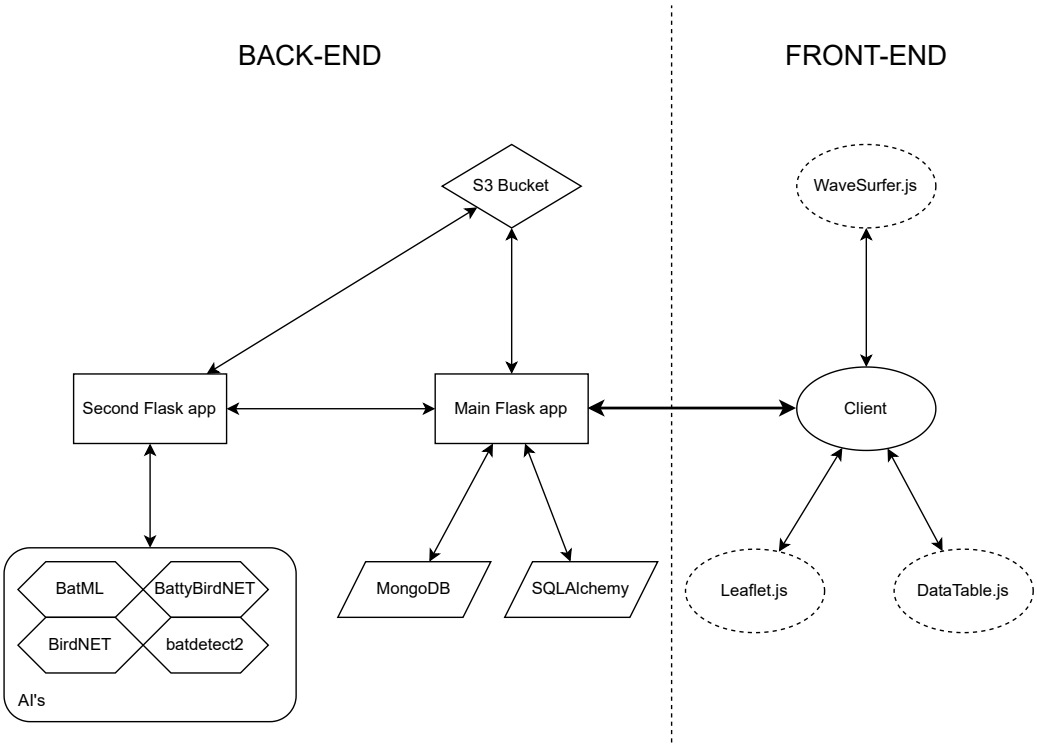


Figure 3.1: Global schema of the web application.
AIs are presented in Chapter 4.
The client is presented in Section 3.2.
Main Flask, Gotham, Bucket, SQLAlchemy and MongoDB are presented in Section 3.3.1.

3.2 Front-End / User Guide

This section presents all the functionalities available on the web application, detailing the interactions between the users and the front-end. Let's go over the layout and functionalities of EchoExplorer:

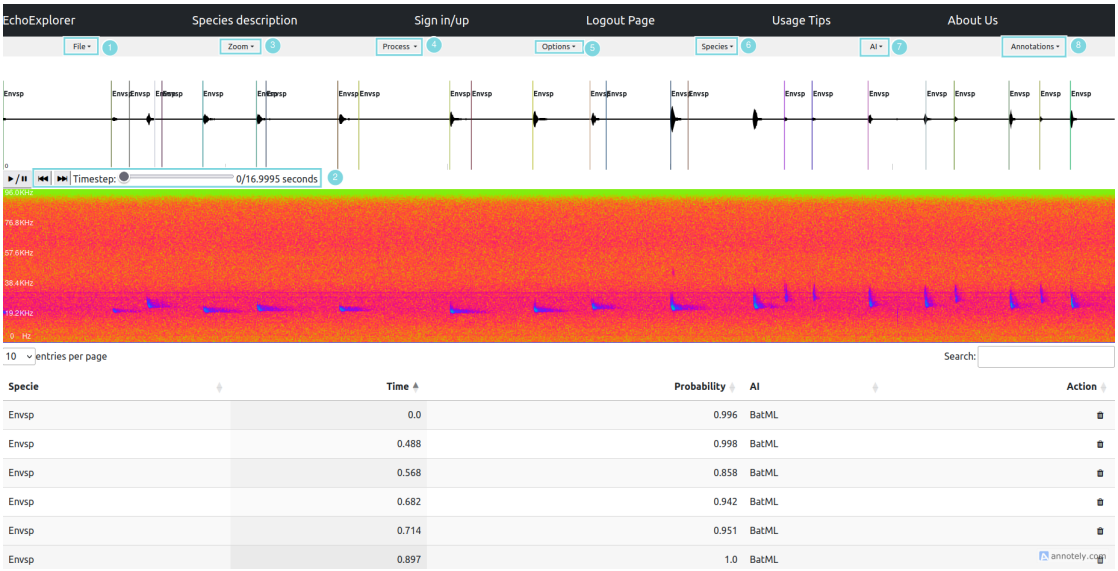


Figure 3.2: Screenshot of EchoExplorer. An audio sample of Nyctalus noctula that has been processed by BatML is shown.

3.2.1 The Waveform

After loading an audio file with the File button, we use the `WaveSurfer.js` library to display all or some part of the waveform of the audio, we also show the spectrogram of the waveform under it. The user can then move along the audio using the arrows or the Timestep slider. In practice, we simply truncate the audio, keeping only the part the user wants to display. Knowing the range of the audio we need to show, the sample rate and the number of channels of the audio, we can select the corresponding bytes to this audio range, without forgetting to include the first bytes corresponding to the WAV header to form a valid sequence of bytes, according to the WAV format.

This is the best way we found since even though the `WaveSurfer.js` library allows moving through waveforms using a slider without needing to reload the audio, which would much simpler, it does not move the spectrogram located under it. We found no way to fix that without getting inside the library's code.

The user can also zoom in and out to show more or less of the waveform using the options in the zoom button. The zoom is also an option that exists within the `WaveSurfer.js` library, but for the same reason as above, using it was impossible without having the spectrogram be unaligned with the waveform, which seemed like a bad idea given its importance for call classification.
An ergonomic improvement could be to allow the user to zoom using the mouse scroll wheel, however loading the waveform takes a bit less than a second which makes that option tedious if not worse than ours.

The spectrogram's maximum frequency can be adjusted through a slider in the Options button.
To create the spectrogram, `WaveSurfer.js` uses the Web Audio API and its object `AudioContext()`. To change the maximum frequency of the spectrogram, the parameter `sampleRate` of `AudioContext()` is then set to the new maximum frequency. But on Firefox, a higher value than 96 kHz seems unsupported, which is far under the maximum range of a bat call.

3.2.2 The Labels

Some points or portions of the Waveform can be labelled with species or custom annotations, we implemented these using the Regions plugin of `WaveSurfer.js`, which allows creating zones on the waveform, and then play or annotate them.
We implemented two kinds of labels:
The first kind are AI-generated. After calling one of the implemented AIs, the timesteps found in the results are labelled with their respective species. AI's labels can get cluttered when a lot of calls are detected, which is one of the reasons why we implemented filters, which we will address later on.

The other kind of label are handmade annotations. The user can create these by clicking and dragging the mouse over some region of the waveform, which can then be adjusted and moved.

When clicking the newly created region, the user can label and annotate the region. After saving, the labels will be displayed on the region and annotations will appear when accessing the annotation menu. Here is how to proceed:

1. Click the region to access annotation form.
2. Select existing or custom annotation and write some note.
3. Save the annotation.

Figure 3.3 illustrates how to annotate a region.

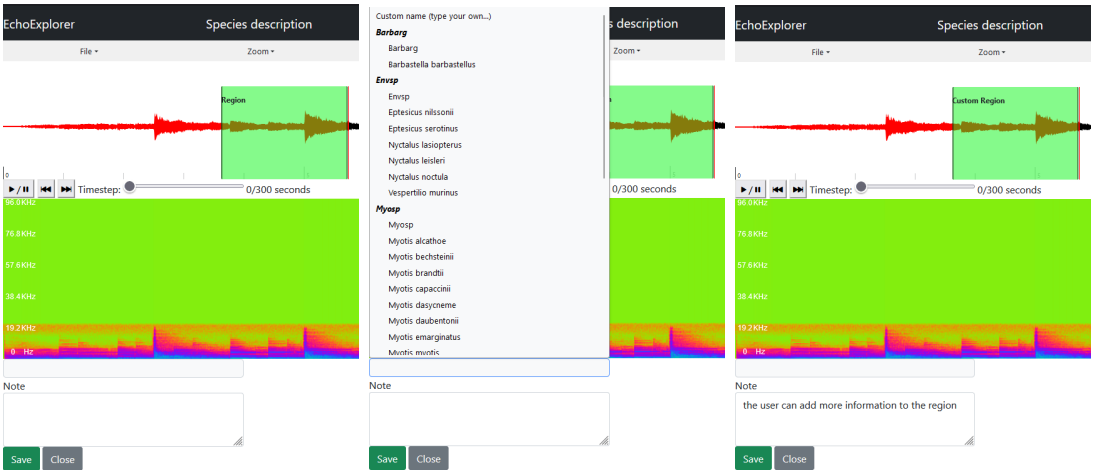


Figure 3.3: How to annotate a region.

3.2.3 The Table

When a label is created on the waveform, be it handmade or provided by an AI, it is added to a table, see Figure 3.4, created with the `DataTables.js` library. The table has 5 columns. Each label (i.e. row) is characterised by its : (i) name (e.g. species); (ii) timestamp; (iii) confidence probability (if AI-generated label); (iv) source (AI-generated or handmade). The last column is a button to delete the label. The user can also delete a label with 'Ctrl'+ click on said label from the waveform. Each column can be sorted, by clicking the column's header.

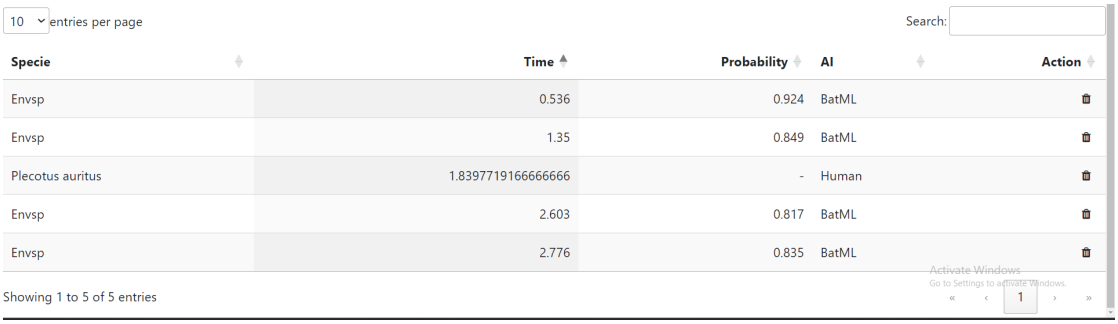


Figure 3.4: The labels table.

If the user clicks on an entry, the minute around its timestamp is displayed on the waveform and the label is highlighted compared to all other ones.
There is also a search box that is implemented by the `DataTables.js` library.

3.2.4 The filters

The user can play around with filters to quickly check for a specific specie or move around the waveform. There are filters on: the species, the source of the label (AI/human), and on the minimal confidence level for predicted labels (respectively (6), (7) and (5) on Figure 3.2).
Both the table and waveform are filtered. This is illustrated on Figure 3.5.

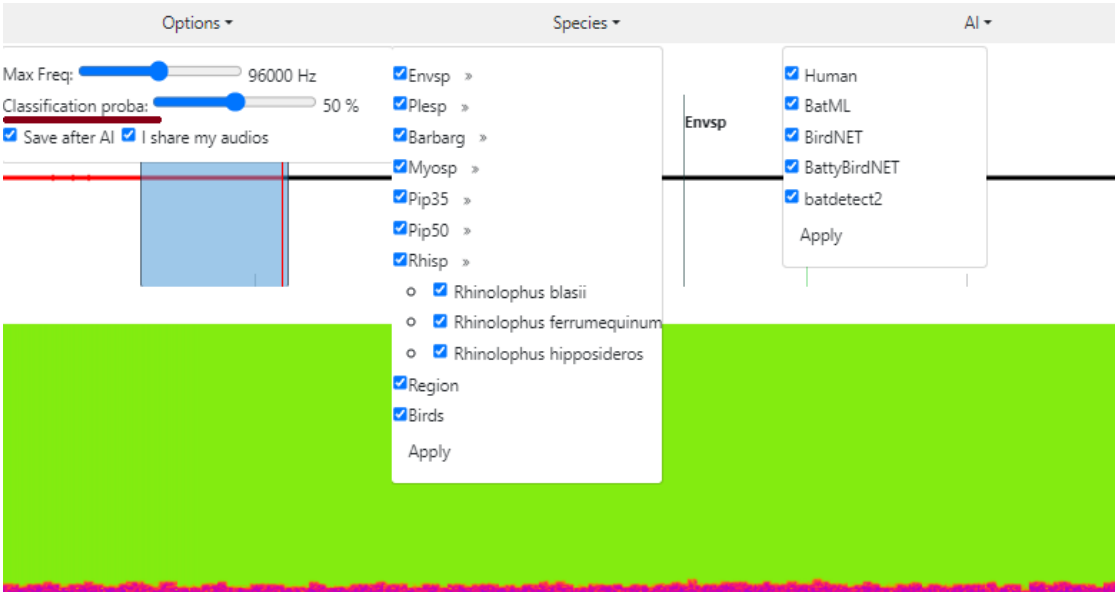


Figure 3.5: The filters.

3.2.5 Process the file with AIs

With the Process button, the user can have the current audio file analysed.
They simply need to click the AI they want. Figure 3.6 shows the list of the available AIs on the web application.
The AIs are presented in Chapters 4 and 6, and their performances are measured in Chapter 5.

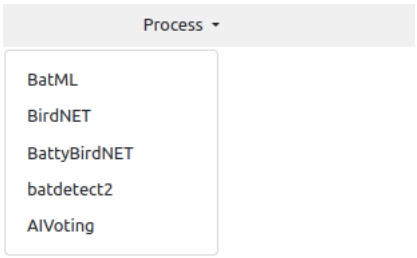


Figure 3.6: Process button.

When the AI has finished processing the file, the timestamps found by the AI are set on the waveform and the table is filled, as shown on Figure 3.2.

3.2.6 Process multiple files at the same time

Users can open and process multiple files at the same time using the File button.

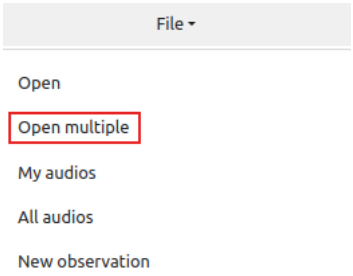


Figure 3.7: Open multiple files.

After the user selects the files he wants, all other buttons and elements available for audio filtering will be disabled, and only the Process button to run the AIs is enabled. After selecting the files, the way to process them with AIs remains the same as previously described in Section 3.2.5. When the analysis is done, the user can find his processed files which have been saved in his section 'My audios' described in Section 3.2.9.

3.2.7 New observation

With the File button, it is shown in Figure 3.8 how a user can open a file and associate it with geo coordinates.
After a user selects his file with Open, they can use the interactive map and click on the location they want to associate their audio file with.

A marker is then set at the position clicked. To cancel, the user can simply click on the marker and no geo coordinates will be saved. The geo coordinates are saved when the user starts the processing of its file. The utility of setting the geo coordinates are explained in Sections 3.2.9 and 3.2.11.

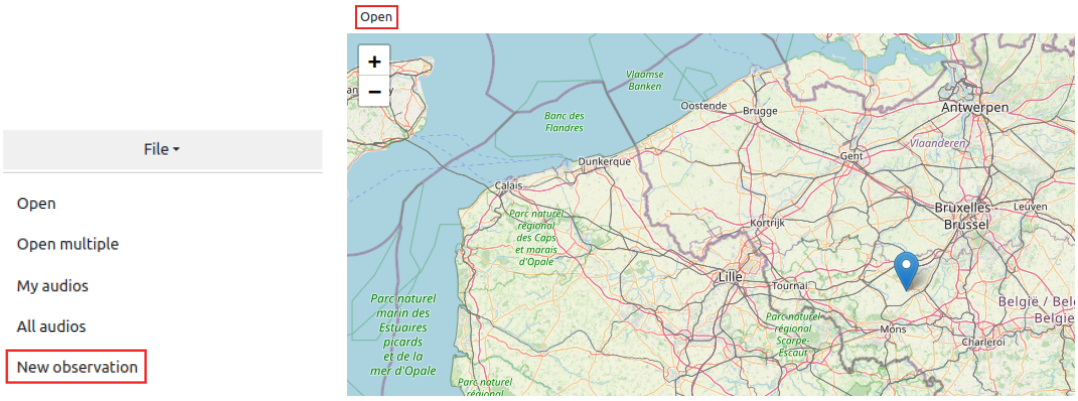


Figure 3.8: New observation button and map to set geo coordinates.

3.2.8 Annotation saving

Each time an AI is called, the current annotations are automatically saved on the server, and the user can retrieve them from its personal space (see Section 3.2.9). If the user adds handmade annotations or modify those given by an AI, they can save them manually through the Annotations button as illustrated in Figure 3.9.

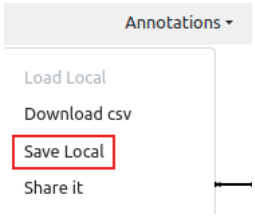


Figure 3.9: Save Local button.

3.2.9 My audios

This section presents the space where a user can manage and see all their saved files. They can access it through the File button as illustrated in Figure 3.10.

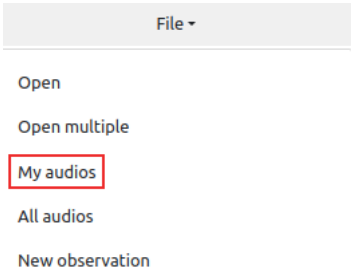


Figure 3.10: My audios button.

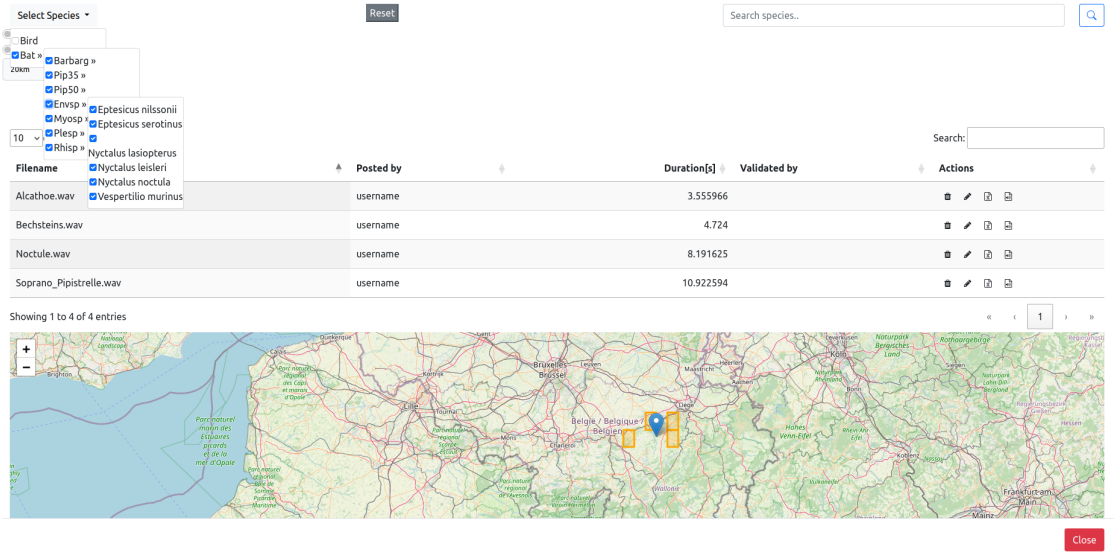


Figure 3.11: 'My audios' space.

The table

The main feature is the table containing all the user's saved files, with some additional information for each file. When the user opens this section, the table shown in Figure 3.11 is then created with the `DataTables.js` library and is filled with all the information sent by the server.

Some action buttons are available to delete or rename the file, to download the audio or the CSV file containing all the annotations made for the audio.

By clicking on a specific row, the audio file and its annotations will be loaded. The longer the file, the longer it will take to load the audio, because of the transfer time between the server and the client-side. A faster way to load an audio with its annotations, which is only possible if the user has the audio locally, is to open the wanted file in the usual way, and then use the button Load Local.

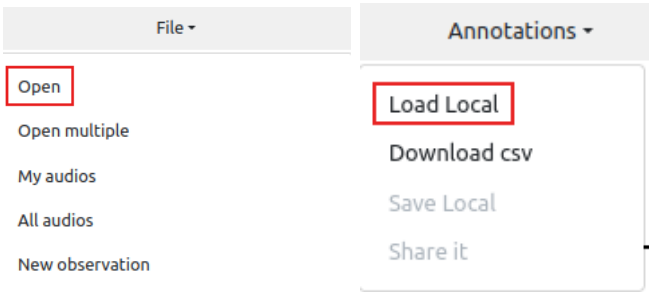


Figure 3.12: Load Local button.

The map

Below the table, there is a map showing where audios were recorded (if provided by the user when analysing the audio, see Section 3.2.7).

The map does not show the exact positions for obvious security reasons regarding animal species. Instead, rectangle areas are displayed, showing that the audio was recorded within some vague area.
In order to do so, we keep the exact locations in the MongoDB database, but when the locations are added to the map, the longitude and latitude are truncated to keep only one decimal.Then, from this truncated position, we create a rectangle by adding 0.1 to the longitude and latitude.

For example, position [51.1234; 7.5678] becomes [51.1; 7.5] and the rectangle [51.1; 7.5], [51.2; 7.5], [51.1; 7.6], [51.2; 7.6] is created. This forms a rectangle because 0.1° does not correspond to the same distance in longitude as in latitude. 0.1° in latitude approximately equals 10km, but the distance in longitude depends on the latitude. If we suppose that we stay in Belgium, 0.1° in longitude approximately equals 15.5km. This might not be the best way to implement this, because we cannot completely decide the size of the rectangles and their shapes differ depending on where they are located. But this was a rather simple way to do it while meeting our requirement that areas should at least be 10 km squares.

The map is implemented with `Leaflet.js` library.

Table filtering

There are several tools to filter the table.

The first one is the map. By clicking on the map, the user will retrieve only the files within a radius of less than 20 km (the amount can be configured by the user), and the table will only list those files.

Next are the species filtering.
One is an auto-completion search, which suggests species to the user after writing some letters. Only species that have at least one occurrence in all the user file list are suggested. After the specie is selected, only files containing this specie are listed in the table.
The second one is the "by taxonomy search", the user can open a recursively nested dropdown of checkboxes with the scientific names and families of the species. Once the wanted species are checked, only the files containing at least one of the species are listed in the table.
This was a good way to address the fluctuation in precision of the different AIs we integrated, some would only return the family of the species while others would give the full scientific name.

Finally, there is also a switch that allows the user to ask to only show validated files. We will go over validated files in the next section.

3.2.10 Sharing an audio

Once a user is done adding annotations, they are able to share their audio to the other users and allow the file to be validated by an expert (described in Section 3.2.11), they can do that by clicking the Share button:

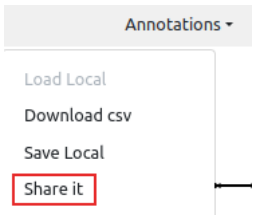


Figure 3.13: Share button.

A user can still modify their already shared annotations if the file is not yet validated by an expert (see Section 3.2.11), otherwise they cannot.

3.2.11 All audios

This section presents the collaborative system implemented in the web application. To feed this system, audio files need to be shared by users as described in Section 3.2.10. This section presents how to access these shared files. It also explains the expert validation system for annotations of shared files.

In the same way as the user can see his files in his own space (see Section 3.2.9), every user can see all the audios shared by any user, through File button as illustrated in Figure 3.14.

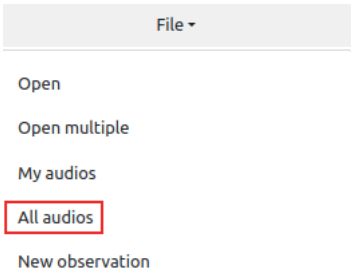


Figure 3.14: All audios button.

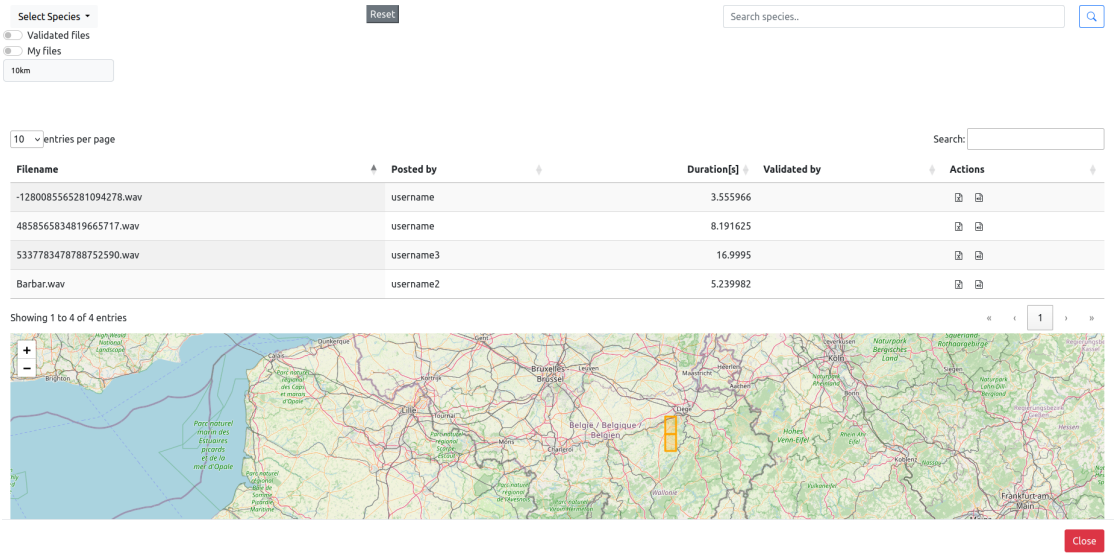


Figure 3.15: 'All audios' space.

The table

When the user opens this 'All audios' space, the table is created in the same way as the table of user's personal files (Section 3.2.9).
For a user, their annotations saved in their personal space (Section 3.2.9) are independent of the annotations saved in the list of all shared audios.
The table is also clickable, and loads any clicked audio with its corresponding annotations, in the same way as the personal files table.

We made the choice that the files shared by the other users appear with their names hashed for privacy reasons. However, the user can still see the actual names of his own shared files (the last row on Figure 3.15). Obviously, the *delete* and *rename* action buttons are no more available.

For privacy reason, the *download* button may not work if the audio's owner has indicated that they do not wish to allow the downloading of their audios. This option is chosen when creating the account on the web application, and can be changed at any moment with the button Options button.

Table filtering

Users can filter this table in the exact same way as the personal files table (see Section 3.2.11), with the auto-completion search, the taxonomy search, the distance search with the map, and the validated files switch. There is also another switch for user to see only their own shared files, to see whether their annotations and status have been modified, for example.

Validating a file

If a user wants to validate files, they have to be classified as an 'expert'.
This for now is granted by us or any administrator of the website. The user can contact us, after which we can grant this additional right to their account, if they have proved by any means that they are a bat expert.

After that step is complete, an expert can load the audio file they want to review by clicking it in the 'All audios' table, or by loading a file from their own computer. The expert can then modify or delete any wrong annotations and add their own. When the expert is done, they can simply click the Validate button, that is available only for expert users:

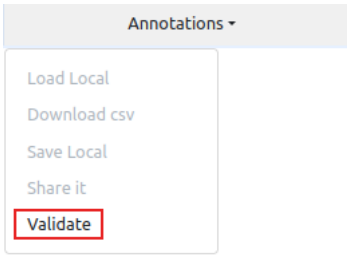


Figure 3.16: Validate button.

Once this is done, the new annotations are saved and the status of the file is updated, with the name of the validator. This only happens in the list of all shared audios, not in the personal file space of the audio's owner.

Figure 3.17: File validated.

If the owner of the audio wants to see the correction made by the expert, they have to navigate to the 'All audios' section, search for their file, and display it by clicking it. If they do modifications on it, they will be able to save (see Section 3.9) it in their personal space, but will not be able to erase the annotations saved by the expert in the 'All audios' section (except if they are also an expert user).

A further implementation idea would be to add some kind of forum to provide a mean of communication between (expert) users.

3.3 Back-End

Now that the interactions between the users and front-end have been presented, this section presents the interactions between the front-end and the back-end to really understand how the functionalities are implemented.

First, some important libraries that we used:

Flask

Flask[33] is a micro web framework in Python, designed to make web development simple. Microframework means that it intentionally lacks most of the expected functionalities in a full-fledged web framework, such as authentication, database abstraction with object relational mapping, input validation/sanitation and web template engine. A microframework facilitates the gestion of HTTP request, such as the receiving, routing and answering of HTTP request. Fortunately, Flask has many extensions to deal with the missing functionalities cited above. Flask is based on several other libraries to allow developers to have a user-friendly experience to create application without paying attention to complex details, like protocol or threads management. These libraries include :

1. **Werkzeug**: is a set of other libraries used to create a WSGI (Web Server Gateway Interface) compatible web application in Python. In other words, it makes the link between the web application and web server.
2. **Jinja2** is a templating language for Python developers, it allows users to write code similar to the Python syntax. It is used to create HTML that is returned to users via HTTP requests.

SQLAlchemy

SQLAlchemy[6] is a SQL toolkit and object-relational mapping (ORM) library for Python. ORM is a technique used to create a 'bridge' between relational databases and an object-oriented language (Python in our case). In this case, SQLAlchemy provides a high level interface allowing users to interact in a very easy way. This way, developers can manage and query a database directly inside a Python script. In that way, SQLAlchemy simplifies the access to the database and allows users to use Python classes in their SQL statements, making it possible to query the database in a Python like way. Finally, users feel like they are simply using a Python object, instead of a database.

Boto3: S3 Bucket

Boto3 [2] is the Amazon Web Services (AWS) Software Development Kit (SDK) for Python, which allows developers to interact with Amazon services, like Amazon S3. Amazon S3 is as object storage service. In our case, we are only interested in S3 bucket, which is a container for objects (in our case, WAV files) stored in Amazon S3. Thus, Boto3 allows developers to interact with buckets to store, retrieve and manage objects. Boto3 enables them to upload files and download files to local storage.

PyMongo

PyMongo [32] is a Python library containing tools to establish connections to MongoDB databases and manage them inside a Python script. MongoDB is a popular NoSQL database which is able to store data, called documents, in a flexible JSON-like format, making it suitable for handling various types of data which could have different structure or change over time.

3.3.1 Schema

Now that the main tools we used to create our web application have been presented, we can dive into more details concerning the schema and our implementation choices.

From the global schema of the web application (Figure 3.1), one can see that the back-end is composed of 2 Flask applications.

Two Flask applications

The main objective of this application is to allow users to easily use bat call detection and identification tools. We want these tools to be ideally as fast as possible to serve the users in the best possible way.

To achieve this objective, we decided to use Gotham (a powerful computer from UCL, with an i7-9800X CPU @ 3.80GHz, 32 GB RAM and an RTX 2080 SUPER GPU on CentOS 8.1.), that will help us to accelerate the process of audio analysis.

Concretely, what we did was to create a first Flask application on the virtual machine used to host the server. This application does everything, except the audio files analysis, including reception of files to be analysed from users.

We then created a second Flask app on Gotham, which the first app will communicate with. In practice, when the first application receives a file from a user, it sends it to the second app via an HTTP POST request. Flask is well suited for handling HTTP requests, as it provides a simple yet powerful interface for defining routes and handling incoming HTTP requests.

3.3.2 Authentication

We implemented an authentication system in the web application for 3 reasons :

- To be able to implement the audio analysis process (described in Section 3.3.3) that uses the username.
- To be able to store annotations that users make (described in Section 3.3.4).
- To be able to distinguish classical and expert users to integrate the collaborative expert validation system (described in Section 3.3.7).

We used SQLAlchemy database to store the username, the password, the flag indicating if they are an expert or not, and the flag indicating if they allow downloading their audios. Figure 3.1 illustrates what looks like the SQLAlchemy table.

Column	**Type**	**Constraints**
id	int	Primary Key
username	varchar	Unique, Not Null
password	varchar	Not Null
isExpert	boolean	Default False, Not Null
shareFile	boolean	Default True, Not Null

Table 3.1: SQLAlchemy User Table.

The password is not really stored in the database, but rather a hashed version of it. See Section 3.4.3 for password hashing.

Given that Flask is very flexible and has many extensions, it is very easy to manage user sessions with Flask-Login [18] and to create forms (used by users to login/register) with Flask-WTF [16], including rendering forms to users using Jinja2.

3.3.3 Process with AIs

This section refers to Section 3.2.5 and explains what happens when user clicks an AI button.

1. **File receiving** :

   When a user submits an audio file, the client-side makes a POST request to the corresponding endpoint of the server, containing the file to analyse as well as which AI to use. Then the main Flask application receives the request and can extract the uploaded file and the name of the AI.

2. **File saving**:

   (i) saving file information in SQLAlchemy database and (ii) saving the audio file itself.

   When the file is received, a unique hash is generated for the filename (from the concatenation of the filename and username, simply with the `hash` function of Python), instead of using the original filename to store it.
   This ensures that each file uploaded by different users has a distinct identifier. This prevents any potential conflict if multiple users upload files with the same original name.

   To manage file information, we created a database table using SQLAlchemy. Each row in the table represents a file and contains the following information:

   - The unique hash name of the file, ensuring its uniqueness in the database.
   - The username who uploaded the file.
   - The original filename of the file on the user's device.
   - The duration
   - The longitude and latitude (if specified, Section 3.2.7)

Column	Type	Constraints
hashName	varchar	Unique
username	varchar	Primary Key, Not Null
originalName	varchar	Primary Key, Not Null
duration	float	Not Null
lng	float	
lat	float	

Table 3.2: SQLAlchemy File Table.

In the database, the hashName is unique as well as the combination of username and originalName.

The audio file itself is not saved on the server, but rather in a S3 bucket. There are several reasons for using S3 bucket. The first one is because the web application needs to keep the audio files to implement the collaborative system, as described at the beginning of Section 3.2.11.
The second one is because this storage system is much more suitable to store huge amounts of files, over storing them locally on a simple machine.

The file is stored in the S3 bucket with the same unique hashed name generated earlier as identifier.

3. **File sending**:

   Since audio files are processed by the second Flask application on Gotham, we had to find a way to send them from the first application.
   Sending audio files of more than a few seconds through HTTP request was impossible because they were too big. Therefore, instead of sending the files to Gotham, we can simply send the filename (the hash name) and the name of the AI to run, through an HTTP request. Then, Gotham can retrieve the hash name and download locally the corresponding file from the S3 bucket.

4. **Audio analysis process**:

   This step is explained in Section 4.5, in the chapter dedicated to the AIs. At the end, a CSV file with the results is written.

5. **Returning the results**:

   When a result CSV file has been written, Gotham sends it back in a JSON format to the main Flask application that is waiting for a response to its POST request. The Flask application then returns the JSON as answer to the initial POST request made on the client-side.

   Unfortunately, if the HTTP connection is lost between the 2 Flask applications, either the main application will wait forever, or the main application will eventually time out and the user will receive an error. This happens because we implemented the audio analysis process in a synchronous way. A further improvement, is to implement it as an asynchronous task (discussed in Section 7.1.3, in the chapter of further improvements).

All these steps are summarised in Figure 3.18.

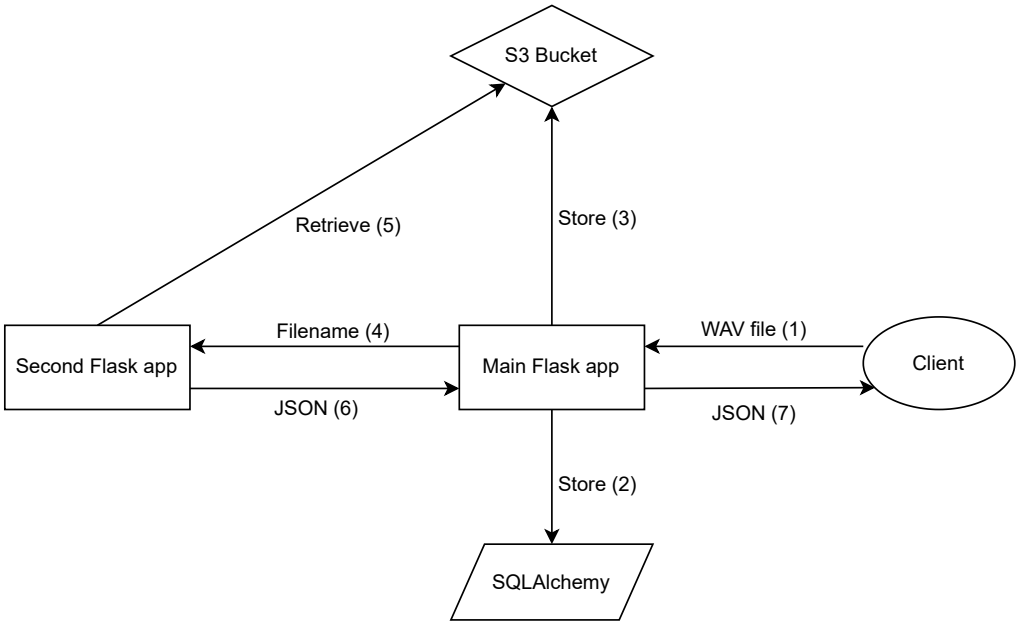


Figure 3.18: Schema of audio analysis process.

3.3.4 Annotations saving

This section refers to the Section 3.2.8 explaining that annotations are automatically saved when an AI is run and that users can save manually their annotations.

We implemented a basic storage system to save user's annotations with MongoDB. In order to be able to use MongoDB in a Python application, we used PyMongo (Section 3.3).

In the web application, we actually use 2 MongoDB collections. The first one is used to store annotations from the 'My audios' space (Section 3.2.9), while the second is used to store the annotations of shared files from the 'All audios' space (Section 3.2.11), its usage is explained in Section 3.3.7.
In this section, it is the first collection that is used.

The annotations are stored in JSON-like format. Each document contains information such as; (i) the hash name, (ii) the original filename, (iii) the username, (iv) the duration, (v) its status (validated or not), (vi) the list of validators, (vii) the latitude and longitude (if provided), and finally, (viii) the list of annotations.

Each annotation is a JSON-like object and is characterised by several attributes such as; (i) the start and end position of the call, (ii) the species, (iii) the confidence probability (if AI-generated), (iv) the note and (v) the source (which AI/handmade).

When an AI is run or when the user manually clicks the button to save their annotations, the client-side sends a POST request to the main Flask application with the list of annotations in JSON format.
The route of the request also contains the username and the filename (either the original name if the user loaded a file from his own machine or the hash name if the user loaded the file from the 'My audios' (Section 3.2.9) or 'All audios' (Section 3.2.11) buttons).

To retrieve the 'filename' and 'originalName' needed to complete the MongoDB document, we used the filename and the username found in the request route.
At this point, we do not know if the filename is a hash name or an original name. So we try a first query to the SQLAlchemy database with `(hashName=filename)`. If it does not find anything, it was an original name, and we can query with `(originalName=filename,username=username)`.

How to determine 'validated' and 'validatedBy' is explained in Section 3.3.7.

Eventually, the document is stored in the collection. Figure 3.19 illustrates the structure of documents stored and what they typically look like.

```
{
    "hashName"     : "14789547138123.wav",
    "originalName": "audioFile.wav"
    "username"     : "username1",
    "duration"     : 25,
    "validated"    : True,
    "validatedBy"  : ["username2",...],
    "annotations"  : [{
                        "start": 1.02,
                        "end": 1.02,
                        "label": "Envsp",
                        "probability":0.87,
                        "note": "this is a note",
                        "ai": "BatML"
                      }
                      ,...],
    "location"     : {
                        "type":"Point",
                        "coordinates"[longitude,latitude]
                      }
}
```

Figure 3.19: Example of document in MongoDB collection.

Given the structure of the data we need to store, relational databases are not well suited. Moreover, we need a database with rich querying capabilities to meet the filtering options described in Section 3.2.9. Therefore, MongoDB is a good choice, in addition to its ease of use.

3.3.5 Sharing an audio

When a user shares an audio, exactly the same thing happens as in the previous section, except that the annotations are saved in the second MongoDB collection.

3.3.6 My audios

This section refers to Section 3.2.9 that explained how users can use their 'My audios' space and manage their saved annotations.

Note: from now on, unless mentioned otherwise, when mentioning the fields that are sent in JSON format, we talk about the fields 'filename', 'originalName', 'username', 'duration', 'validated' and 'validatedBy'.

The table

When the user opens this space, the client-side makes a request to the main Flask application, that will then query from the first MongoDB collection all the documents of the user using his username.

In a JSON format, the fields of each document are returned to the client-side to fill in the table.

The *delete*, *rename* and *csv* action buttons have interaction with the MongoDB collection, while *download* action button interacts with the S3 bucket. All these buttons use as filename the hash name that is not shown in the table.

1. the *delete* button deletes the document with `(hashName=filename)`. If the document has not yet been shared, it is also deleted from the S3 bucket and SQLAlchemy database to avoid wasting space.
2. the *rename* button changes the field 'originalName', the name actually shown in the table, of document with `(hashName=filename)`.
3. the *csv* button queries the field "annotations" of document with `(hashName=filename)`, and download the list of annotations as a CSV file on the user's machine.
4. the *download* button retrieves from the S3 bucket the audio with `(hashName=filename)` and downloads it on the user's machine.

Table interaction

By clicking on a specific row, the audio file and its annotations will be loaded. Concretely, this sends a request to the server, containing the hash name and the username. The server can then retrieve the file from the S3 bucket as well as the annotations by querying the MongoDB collection with `(filename=hashName,username=username)`. After that, everything is returned to the user and can be displayed.

Table filtering

The auto-completion search and the taxonomy search work in the same way. A list of species (or one species for auto-completion search) is sent to the server.
The MongoDB collection is then queried to retrieve all documents with at least one occurrence of any of the species in the list. Then, in a JSON format the fields of each document are returned to the client-side to fill in the table.

Figure 3.20 shows what a typical query looks like.

```
{
   "annotations": {
      "$elemMatch": {
         "$or": [{"label": "Envsp"},
                 {"label": "Barbarg"},
                 ...
                ]
       }
   }
}
```

Figure 3.20: MongoDB query on the species.

Table filtering with the map

When the map is clicked, it fills in the table with files located within the radius of the clicked position.

When the user clicks the map, the client sends a request with the longitude and latitude of the clicked position, as well as the parameter radius, which is configurable by the user.
The server then uses MongoDB geospatial spatial query [19] to retrieve only the files within the radius. In a JSON format the fields of each document are returned to the client-side to fill in the table.

Figure 3.21 shows what a typical query looks like.

```
{
   "location": {
      "$near": {
         "$geometry": {'type': 'Point', 'coordinates': [lng,lat]},
         "$minDistance": 0,
         "$maxDistance": radius
      }
   }
}
```

Figure 3.21: MongoDB geospatial query.

3.3.7 All audios

This section refers to Section 3.2.11 that explained how user can use their 'All audios' space.

Unless mentioned otherwise, this section functions exactly the same as the 'My audios' section 3.3.6 except that the MongoDB collection used is the second one. We have made this choice so that annotations on shared files and user's personal files can be clearly distinguished. Therefore, when an expert modifies the annotation of a shared file, it does not impact the audio owner's personal annotations.

Validating a file

When an expert validates a file, it saves the annotations in the MongoDB collection of shared files. The status of the document is set to `True`, and the name of the expert is added to the list of validators of this document. Other experts can still modify any validated files.

When a user wants to save their corrected file in their 'My audios' space, the server checks if the file is in the MongoDB collection of shared audios. If so, when the user saves it, the status of their file is set to True and the username of the expert is added to the list of validators.

Example of combined query

For both 'My audios' and 'All audios' spaces, Figure 3.22 shows an example of MongoDB query that combines all filters available for the users.

```
{"$and$":[{
              "location": {
                 "$near": {
                    "$geometry": {'type': 'Point', 'coordinates': [lng,lat]},
                    "$minDistance": 0,
                    "$maxDistance": radius
                 }
              }
           },
           {
              "annotations": {
                 "$elemMatch": {
                    "$or": [{"label": "Envsp"},
                            {"label": "Barbarg"},
                            ...
                           ]
                 }
              }
           },
           {
              "validated":True
           },
           {
              "username":username
           }
         ]
}
```

Figure 3.22: MongoDB combined query.

3.4 Security

In any modern web application, ensuring robust security is a really important task from the user's point of view, but also as the application manager.
Users do not want sensitive data to be divulged to other users, and especially not to hackers. Neither do they want to be tricked by a hacker to perform involuntary actions.

3.4.1 File verification

As one of the main objective of this application is the analysis of audio files, we need to be really careful concerning any uploaded files.
First, we need to sanitise and secure the filenames themselves. To do that, we used the function `secure_filename()`, from `werkzeug` library [35], this ensures that the filenames are transformed into safer versions, removing or replacing all characters that could be used for malicious purposes, such as directory traversal attacks.
Otherwise, it could lead to the file being saved in an unintended directory, potentially compromising the server's security. For example, "../../../../../etc/passwd" is not a secured filename.

After sanitising the filename, we need to check if the content of the file is not malicious.
To begin with, the server only accepts files with .wav extension. A hacker could simply upload a malicious file with .wav extension to bypass this very simple test. This prompts us to also check if the first few bytes match the header format of every WAV file (Section 2.3).

3.4.2 Secured authentication form

For our application to work properly, we need to be able to identify users.
The users can sign up themselves via a register form, providing a valid username and password.
Since everything that comes from the user is a potential risk for an attack, we must be very careful about how we handle customer input via this form.

With Flask-WTF [16], we were able to provide more secured sign in/up forms.
Flask-WTF has built-in functions to sanitise user's input, and it also provides built-in CSRF protection, which is explained in Section 3.4.7.

3.4.3 Hashed password

Hashing passwords in web application is an important aspect to protect user credentials.
In the case of a security breach, even if a hacker has access to these information, they will not be able to get any passwords if only the hashed passwords are stored. Indeed, hash functions are designed to be "one-way", meaning that it is infeasible to retrieve the password based on their hashed value. Moreover, hashing a password also provides protection against people with access to the database, or unauthorised individuals that would gain access to the database.

Concretely, we used `Bcrypt` from the Flask-Bcrypt extension of Flask [13]. Which provides bcrypt hashing utilities.

We use bcrypt [31] because it is well suited for passwords protection purpose, thanks to these 2 key features:

1. Slow runtime: bcrypt is a slow algorithm that takes time to create password hashes and decrypt them. Thus, it significantly increases the time taken by a hacker using brute-force methods.
2. Usage of salt[1]: adding a random data and hashing it together with the password helps generate unique password hashes resistant to rainbow attacks (when hacker compares stolen hashed password with many pre-computed password hashes).

3.4.4 Request limiter

To enhance security against brute-force attacks, we added a request limiter for log in requests, with a rate of 5 attempts per minute.
We also added the same limiter for registering request, preventing from malicious people that would create a lot of accounts and overload the server.
Finally, we set the limiter of 5 attempts per minute for audio processing request too. This prevents malicious users from clicking many times on buttons to analyse files and overload the client.
Users are still able to analyse more than 5 audios at once, as in Section 3.2.6, because this is considered as one request.

This was very easily implemented thanks to Flask-Limiter [15], an extension of Flask.

[1] a salt is a random piece of data that is added to each password before it is hashed

3.4.5 SQL injections

A SQL injection is a technique that might leak confidential information or destroy the database if user's input is not correctly handled.
A naive approach to deal with queries on databases inside a user's input, would be to insert the user's input into a parameterised query.

Here is an example[2],

"SELECT * FROM Users WHERE UserId = " + userInput

if user input is "105 OR 1=1", the query would look like this,

"SELECT * FROM Users WHERE UserId = 105 OR 1=1;"

and would return all rows from the 'Users' table, since "OR 1=1" is always True.

To counter this kind of attack, we must use ORM libraries with built-in protection against SQL injection that allows to interact safely with a SQL database. In our case, we decided to use SQLAlchemy.

3.4.6 Secured communication

We use HTTPS to encrypt data transmitted between the user and server to prevent man-in-the-middle attacks.

3.4.7 CSRF protection

Cross site request forgery (CSRF), is an attack that tricks a web browser into executing an unwanted action in an application to which a user is logged in.
CSRF attacks typically happen using an email or a link that tricks the victim into sending a forged request to a server.
If the user is authenticated in the application at the time of the attack, it is impossible to know whether the request is desired or not.

To prevent this attack, we must use a CSRF token to validate the request origin when dealing with POST requests. Moreover, we must ensure that no GET request perform changes on the server because this protection cannot be used with GET requests.

[2] https://www.w3schools.com/sql/sql_injection.asp

Concretely, when a user logs in, the server will generate a CSRF token linked to the user's session and send it back to the user. With each POST request, the server checks whether the token received with the request corresponds to the token linked to the user's session.

In practice, we use the `flask_wtf.csrf` module. We added the CSRF token in the login form, as well as in all other POST requests header made in the JavaScript code. And that is it, everything else is handled by Flask itself.

Chapter 4

AIs

This chapter briefly presents each AI available on EchoExplorer. We will then explain what format must be followed and the steps to take to integrate a new AI to the web application.

4.1 BatML

BatML [7] is a Master's thesis project that was led by Mélanie Bauveois and Lucile Dierkx at Université Catholique de Louvain during the academic year 2020-2021. This project follows a previous Master's thesis, Batmen [17].
Batmen's objective was to perform multi-class detection, building on the Batdetective project [25] that simply performed bat call detection.

BatML's objective was to perform multi-label classification of bat calls, with Batmen as starting point.
The purpose of multi-label classification is to detect every bat call positions in an audio and then identify the bat species that emitted the call. While multi-class classification is not intended to detect the calls positions, but only predict the bat specie that is the most likely to be in the audio.

Dataset

To perform multi-label classification, the call positions are required, unfortunately, the dataset provided by Plecotus, the bat department of Natagora [28], only contains the bat groups present in the files, and not the call positions.

However, if a file contains only one group, it could still be used, as each call detected by Batdetective can be assigned to this group.

They therefore discarded all the files containing multiple bat groups, since they could not decide which call was emitted by which group. Then, new recordings containing superpositions of calls between pairs of groups were artificially created.

Table 4.1 lists the 23 bat species found in Belgium, and their associated groups used to train the classifier.

Scientific name	Common name	Group
Barbastella barbastellus	Western barbastelle	Barbarg
Eptesicus serotinus	Serotine bat	Envsp
Myotis alcathoe	Alcathoe bat	Myosp
Myotis bechsteinii	Bechstein's bat	Myosp
Myotis brandtii	Brandt's bat	Myosp
Myotis dasycneme	Pond bat	Myosp
Myotis daubentoni	Daubenton's bat	Myosp
Myotis emarginatus	Geoffroy's bat	Myosp
Myotis myotis	Greater mouse-eared bat	Myosp
Myotis mystacinus	Whiskered bat	Myosp
Myotis nattereri	Natterer's bat	Myosp
Nyctalus lasiopterus	Greater noctule bat	Envsp
Nyctalus leisleri	Leisler's noctule	Envsp
Nyctalus noctula	Common noctule	Envsp
Pipistrellus kuhlii	Kuhl's pipistrelle	Pip35
Pipistrellus nathusii	Nathusius's pipistrelle	Pip35
Pipistrellus pipistrellus	Common pipistrelle	Pip50
Pipistrellus pygmaeus	Soprano pipistrelle	Pip50
Plecotus auritus	Brown long-eared bat	Plesp
Plecotus austriacus	Grey long-eared bat	Plesp
Rhinolophus ferrumequinum	Greater horseshoe bat	Rhisp
Rhinolophus hipposideros	Lesser horseshoe bat	Rhisp
Vespertilio murinus	Parti-coloured bat	Envsp

Table 4.1: Scientific and common names of bats in Belgium, as well as their group.

The performances reported are the following:

	Accuracy	Precision	Recall	F1-score
Detect	0.99	0.86	0.69	0.77
Classify	0.93	0.75	0.79	0.76
Global	0.99	0.72	0.57	0.61

Source: BatML [7].
Table 4.2: BatML performances on test set.

4.2 BatDetect2

BatDetect2 [3], published in 2022, is the continuation of the Batdetective project which was used to create Batmen and BatML.
BatDetect2's objective is to perform multi-label detection, just like BatML does.

BatDetect2 was trained on 4 datasets coming from the United-Kingdom, Mexico, Australia and Brazil.
In our web application, we use the classifier trained with bats from the UK, containing 17 bat species, listed in Table 4.3.

Species	Num Train Calls	Num Test Calls
Barbastella barbastellus	468	575
Eptesicus serotinus	403	2182
Myotis alcathoe	374	504
Myotis bechsteinii	241	629
Myotis brandtii	351	1590
Myotis daubentonii	3998	2371
Myotis mystacinus	1378	1436
Myotis nattereri	2610	102
Nyctalus leisleri	695	446
Nyctalus noctula	209	200
Pipistrellus nathusii	1460	0
Pipistrellus pipistrellus	868	1030
Pipistrellus pygmaeus	1461	1106
Plecotus auritus	528	582
Plecotus austriacus	331	536
Rhinolophus ferrumequinum	717	1488
Rhinolophus hipposideros	722	1571

Source: BatDetect2 [3].
Table 4.3: Species count in train and test set of BatDetect2.

4.3 BirdNET

BirdNET [22] is a model performing multi-label classification on bird species. It has been trained to be capable of identifying 984 North American and European bird species.

4.4 BattyBirdNET

BattyBirdNET [38] is built upon the work of the BirdNET [22] project, making it suitable for bat call detection and classification.

BattyBirdNET has been trained with different datasets coming from Europe, USA, United-Kingdom, Scotland and Bavaria. The Bavarian dataset is available here. This data is a collection taken from Xeno-Canto [37], chirovox, animal sound library berlin as well as from individuals (R. Zinck (GER), K. Richards (UK)).

Scientific name	Common name	Group
Eptesicus nilssonii	Northern bat	Envsp
Hypsugo savii	Savis pipistrelle	Pip35
Myotis capaccinii	Long-fingered bat	Myosp
Rhinolophus blasii	Blasius horseshoe bat	Rhisp
Plecotus spec.	Long eared bat	Plesp
Tadarida teniotis	European free-tailed bat	Other
Miniopterus schreibersii	Common bent-wing bat	Other

Source: BattyBirdNET github [38].
Table 4.4: Additional bats covered by the BattyBirdNET classifier not listed by BatML's Table 4.1.

4.5 How to add an AI to the web application

The web application is intended to be very modular and is not constrained to only use AIs specialised in bat detection and classification. It should be easy to add any new AI performing animal call identification to EchoExplorer.
The objective of this section is then to present how to do that, as well as how the already integrated AIs work.

4.5.1 AI requirements

Firstly, the AI itself must meet certain criteria.

The input

The way the AI receive its inputs should be the following; All files given as input to the AI must be stored in an input folder. This folder is the directory in which the script running the AI will look for audio files to process.

The AI should handle multiple input files.

The output

The way the AI outputs the analysis of the files is the following; The AI must write to a single CSV file which lists all the calls detected and identified for each input file.
It does not matter if a separate file is created for each input file before grouping all of them into a main CSV, as long as there is this unique result file containing all the calls identified.

Filename	Start	End	Scientific name	Confidence

Table 4.5: CSV output file format.

The format of the unique output CSV file is shown in table 4.5. Each row corresponds to one identified call.
The file must have 5 columns. (i) the file name, (ii) the starting position of the call, (iii) the end position of the call (same as starting position if not provided by the AI), (iv) the scientific name identified in the call, (v) the confidence probability about the species identified.

The file must be stored in an output folder.

Running the AI

The AI must be callable from a Python script. And the command to call it must at least specify the path to the input and output folders. Additionally, the command must handle one more parameter, the username, to ensure its integration into the web application.

4.5.2 AI integration

Front-end

There are 2 minimal modifications to bring to the front-end side.

1. add the HTML code to create a new button in the main HTML file named index.html
2. link the new button with the code routine to execute when any AI button is pushed. Choose a name for this new AI, and use this name in both of the Flask applications.

Back-end

The only modification to bring to the main Flask application, is a very simple computation of the approximated time needed to fully process the client's request. This approximation is based on the total duration and the total size of its files.

The total duration is used to approximate the time needed by the AI to analyse all the files.
For a single file, at equivalent duration but higher sample rate and number of channels, the time needed by the AI to process the file is a little bit longer, but this can be negligible.

The total size of the files is used to approximate the transfer time from the client-side to the server-side. This time is heavily dependent on the client's network quality. Only the time taken by the AI itself must be measured, since the transfer time is independent of the AI processing.
We ran a test multiple times to have a more reliable time transfer approximation. The time needed to analyse the file did not change from a run to another.

To calculate the numbers in Table 4.6, we sent an audio file of 1 hour. We measured the time taken for the Gotham application to receive the request and then the time taken by each AI to analyse the file.

	Duration[s]
BatML	$\frac{duration}{3}$
BirdNET	$\frac{duration}{54}$
BattyBirdNET	$\frac{duration}{5.5}$
BatDetect2	$\frac{duration}{10}$
New AI	...

Table 4.6: Time approximation for files analysis.

The integration mostly takes place in the Flask application that is run on Gotham. To begin, just put the folder with the new AI at the same level as the other AIs :

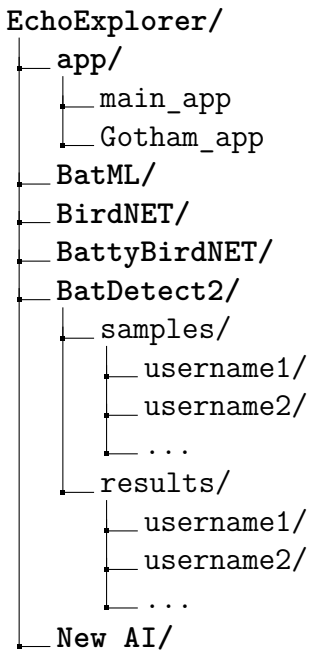


Figure 4.1: Project Directory Structure.

Then, in the Gotham Flask application, add the code to handle this new AI.

1. define and create a unique input folder **based on the username**.
2. define and create the output folder.

3. download locally, from the S3 bucket, all the files sent by the client into this folder.
4. start the AI, for example with the function `subprocess.run()` and provide at least the input and output folders as well as the username. The username is also used to create a unique name for the resulting CSV file to be able to distinguish it from those of other users files.
5. define the path of the CSV file that contains the results.
6. delete all the audios sent by the client that have been saved locally, as well as any potential intermediary result files used to create the main one.

4.5.3 AI analysis routine

When the hash name and the name of AI are received by Gotham from the main Flask application, Gotham downloads locally the audio file corresponding to the hash name from the S3 bucket, in the `AI/samples/username/` folder of the given AI and user making the request.
Additionally, the `AI/results/username/` folder is also created to store the potential intermediate and final result files.

Now, the AI can be started as explained in Section 4.5.1, with this kind of command:

```
python3 ai_script samples/username results/username username
```

Chapter 5

Experiments

This chapter presents all the experiments we did to compare, mainly, the AI performing bats detection.

Every AI used in this web application performs multi-label classification. This means that for a given audio file, it outputs the list of detected calls and their positions, and then associates each call with a bat group/species. Different group/species can be outputted for a single file. As opposed to multi-class classification, which outputs only one bat group/species for the whole file. However, for this work, we do not have access to a dataset suitable for multi-label classification task. Nevertheless, we still have access to datasets usable to measure and compare the performances of the AIs, for multi-class classification task.

5.1 Performance metrics

This section presents an overview of the main metrics that are used to measure the performances for a given classification task in machine learning.

Confusion matrix

The confusion matrix is not a metric, but rather a tool that eases the computation of the next metrics, presented right after this.

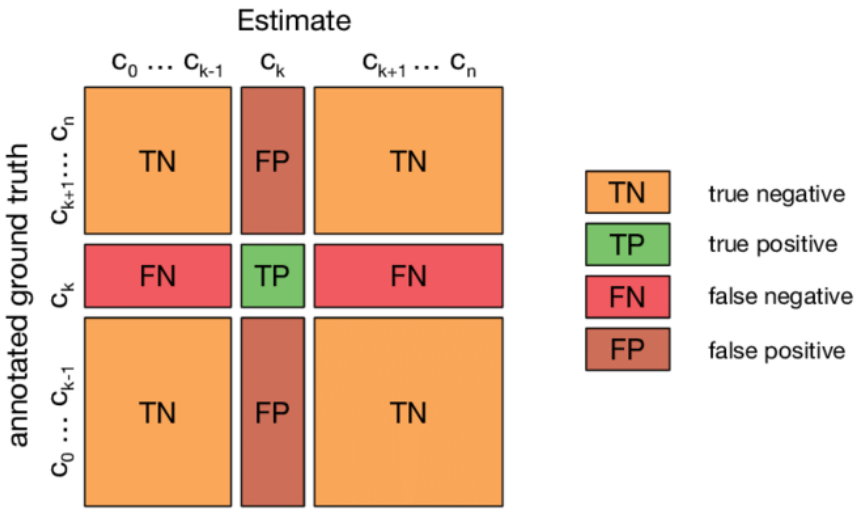


Figure 5.1: Multi-class confusion matrix.

True Positive (**TP**) is the number of examples for which the predicted class corresponds to the considered class.

True Negative (**TN**) is the number of examples that are predicted as not belonging to the considered class and that do not belong to that class.

False Positive (**FP**) is the number of examples that are predicted as belonging to the considered class, while they do not belong to that class.

False Negative (**FN**) is the number of examples that are predicted as not belonging to the considered class, while they do belong to that class.

Precision

Precision [21] is the ratio between the number of correctly classified examples for a given class and all examples assigned (by the model) to this class. Thus, *precision* $\in [0, 1]$, and is defined such as:

$$Precision = \frac{TP}{TP + FP}$$

Precision can be interpreted as the ability of the model to avoid false positive predictions. A higher precision value indicates fewer false positives.

Recall

Recall [21], also known as sensitivity, is the ratio between the number of correctly classified examples for a given and all actual examples assigned to this class. Thus, $recall \in [0, 1]$, and is defined such as:

$$Recall = \frac{TP}{TP + FN}$$

Recall can be interpreted as the ability of the model to avoid false negative predictions. A higher recall value indicates fewer false negative predictions.

F1-score

F1-score combines both *precision* and *recall*. It computes their harmonic mean. Thus, $F1\text{-}score \in [0, 1]$, and is defined such as:

$$F1\text{-}score = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

Multi-class

The metrics presented above are the versions for binary classification. To adapt them to multi-class classification [21], the classical way is to take the average of the metric evaluated for each class.

$$Precision = \sum_{i=1}^{i=K} Precision_i = \sum_{i=1}^{i=K} \frac{TP_i}{TP_i + FP_i}$$

$$Recall = \sum_{i=1}^{i=K} Recall_i = \sum_{i=1}^{i=K} \frac{TP_i}{TP_i + FN_i}$$

$$F1\text{-}score = \sum_{i=1}^{i=K} F1\text{-}score_i = 2 \times \sum_{i=1}^{i=K} \frac{Precision_i \times Recall_i}{Precision_i + Recall_i}$$

5.2 Datasets

This section presents the two datasets used to evaluate the performances.

Lauzelle dataset

The first dataset is a set of files collected and labelled by Yves Laurent. The audio files come from the 'Bois de Lauzelle' in Louvain-la-Neuve. For each file, one bat species is associated with. However, it's not impossible that other species not mentioned may be in the file. We also kept audios with birds, parasite and empty audios in the dataset to check whether the AIs don't say there are bats in it. Indeed, predicting when there is nothing is just as important as predicting when there are bats.

Xeno-Canto dataset

To enrich our tests, we decided to create a second dataset using Xeno-Canto [37]. Xeno-Canto is a website where animals enthusiasts can share wildlife sound recordings from all over the world. One can listen, download and explore every sound recordings in the collection. After a sound recording is posted, it can be validated by other expert users. Thanks to this feature, we were able to download validated audios of multiple bat species and create our own dataset of bats. Compared to the dataset provided by Yves Laurent, Xeno-Canto ensures that only one species is in a recording.

From Xeno-Canto website : « *What is a 'soundscape' recording?*
In 2013, we added the ability to upload 'soundscape' recordings on Xeno-Canto. The intention is to allow people to share longer recordings where more than a single species is vocalising. The prototypical example is a recording of a dawn chorus, where many species are vocalising together. In such cases, attempting to extract a single-species portion of the recording will often diminish its value.
Soundscape recordings should generally be at least a minute long, and have two or more species identified in the background species list. Occasional uploads of non-avian species are usually acceptable, but we discourage large numbers of these recordings. Eventually, we may add proper taxonomic support to allow people to share vocalisations of different classes of animals, but for now the focus is on birds.»

To proceed in the same way as the Lauzelle dataset, we also decided to download audios files in which there is no bat. Thus, we have downloaded audios of house cricket and field cricket. We chose these insects because they also emit ultrasounds like bats, to challenge the AIs and make sure it's not too easy for them to detect free-bats audios.

We used `scikit-maad` Python library to download audios of Xeno-Canto from a Python script, which is much more practical than doing it manually.

Datasets details

Table 5.1 reports the number of files containing each bat species for both datasets. Unfortunately, there are many bat species that are not in the Lauzelle dataset plus some other species with very few occurrences. For the Xeno-Canto dataset, all species are represented, but there are still some of them with very few occurrences. This will be important to take into account when analysing and interpreting the performances.

Species	Lauzelle	Xeno-Canto
empty	152	124
Barbastella barbastellus	0	54
Eptesicus nilssonii	0	71
Eptesicus serotinus	248	34
Myotis alcathoe	0	1
Myotis bechsteinii	0	5
Myotis brandtii	2	8
Myotis capaccinii	0	1
Myotis dasycneme	0	5
Myotis daubentonii	19	38
Myotis emarginatus	1	3
Myotis myotis	52	16
Myotis mystacinus	14	5
Myotis nattereri	8	21
Nyctalus lasiopterus	0	4
Nyctalus leisleri	27	88
Nyctalus noctula	29	95
Pipistrellus kuhlii	2	39
Pipistrellus nathusii	16	46
Pipistrellus pipistrellus	313	100
Pipistrellus pygmaeus	34	66
Rhinolophus ferrumequinum	0	8
Rhinolophus hipposideros	0	11
Vespertilio murinus	0	34
Plecotus auritus	3	46
Plecotus austriacus	3	5

Table 5.1: Species count for Lauzelle and Xeno-Canto datasets.

One of the AI available on the web application predicts bat species groups rather than species, this is BatML. To be able to compare it with the two other AIs, we had to convert the species datasets into species groups datasets. Moreover, to be able to compare these new datasets with the predictions made by an AI, each predicted species needs to be mapped to its corresponding species group.
Table 5.2 reports the number of files containing each bat species group. In addition to being able to compare BatML with the other 2 AIs, it is also interesting to use the species groups to assess the performances because an AI may be wrong about one species because it is very similar to another one of the same group, and in this case the AI is not wrong. Doing it this way, it also reduces the bias in the results due to the fact that certain species occur only a few times in the datasets. But the Lauzelle dataset has still two empty groups and two poorly represented groups, because these species are not present in Bois de Lauzelle. While the Xeno-Canto dataset has only one group with few occurrences.

Species	Lauzelle	Xeno-Canto
empty	152	124
Barbarg	0	54
Envsp	302	326
Myosp	93	103
Pip35	16	85
Pip50	318	166
Plesp	7	51
Rhisp	0	19

Table 5.2: Species groups count for Lauzelle and Xeno-Canto datasets.

5.3 Performances

To gather the predictions of each file, we used the confusion matrix. The rows of the matrix are the actual classes and the columns are the predicted classes. To fill in the confusion matrix, we iterate over each file and their predictions.

The metrics

To evaluate the performances of the 3 AIs, 3 metrics are used:

1. *Precision* is used to measure the ability of an AI to avoid predicting a wrong class. Indeed, the more an AI makes wrong predictions for a given class, the higher is FP, and the lower *precision* is.

2. *Recall* interprets the ability of the AI to correctly classify all occurrences of a given class. Indeed, the more an AI misses occurrences of a class, the higher FN is, and the lower *recall* is.

3. *F1-score* is used to calculate a trade-off between the two first ones.

The metric that most significantly influences our comparison is recall. A higher recall indicates that the AI achieve to identify a large proportion of the true species. Due to the precision-recall trade-off, increasing recall can lead to a decrease in precision. However, in our opinion, achieving a higher recall is more important than higher precision in the context of bat call identification. To help chiropterologists to validate the AI's predictions, it is easier for them to discard or correct wrong predictions than to manually search through the entire recording to find species that the model missed.

Lauzelle confusion matrix

With the Lauzelle dataset, for a file to be correctly classified, the actual species associated with the file must be predicted at least one time by the AI, otherwise, the file is wrongly classified. If a file is empty, it is correctly classified only if the AI predicts nothing. If a file is correctly classified, in the confusion matrix, the element `(true class, true class)` is incremented by one.

However, because of the assumption saying that it is not impossible that other non mentioned species may be in the file, we cannot increment the element `(true class, predicted class)` for each predicted class that was not mentioned by Yves Laurent. In this way, the confusion matrix ends up being diagonal. Because of that, only the *recall* metric can be calculated, with TP the element on the diagonal and $TP + FN$ is known thanks to Tables 5.1 and 5.2. *Precision* cannot be calculated because there is no FP in the matrix, so *F1-score* cannot be calculated either.

Xeno-Canto confusion matrix

With the Xeno-Canto dataset, the diagonal elements `(true class, true class)` are incremented in the exact same way as with the Lauzelle dataset.

The difference is in the calculation of FP. Since we know that only one species can be in a given file, each other predicted species than the right one, increments the element `(true class, predicted class)`. Thanks to that, *precision* can also be calculated, and thus *F1-score* too.

5.3.1 Results

Firstly, the comparison between the 2 AIs using bats species rather than species group, which are BattyBirdNET and BatDetect2, is presented. Then, using the species group, the comparison with the 3 AIs is presented.

Bats species

Species	Lauzelle		Xeno-Canto					
	BattyBirdNET	BatDetect2	BattyBirdNET			BatDetect2		
	recall	recall	precision	recall	F1-score	precision	recall	F1-score
empty	0.63	0.94	0.85	0.90	0.88	0.79	0.81	0.80
Barbastella barbastellus	/	/	0.75	0.92	0.83	0.51	0.96	0.67
Eptesicus nilssonii	/	/	0.43	0.69	0.53	/	/	/
Eptesicus serotinus	0.81	0.92	0.60	0.79	0.68	0.41	1	0.59
Myotis alcathoe	/	/	0	0	0	0.5	1	0.67
Myotis bechsteinii	/	/	0.17	0.2	0.18	0.25	0.4	0.30
Myotis brandtii	0	0	0	0	0	0.28	0.63	0.38
Myotis capaccinii	/	/	0	0	0	/	/	/
Myotis dasycneme	/	/	0.57	0.8	0.67	/	/	/
Myotis daubentonii	0.74	1	0.38	0.63	0.47	0.36	0.95	0.53
Myotis emarginatus	0	/	0.75	1	0.86	/	/	/
Myotis myotis	0	/	0.38	0.81	0.52	/	/	/
Myotis mystacinus	0.86	0.93	0.44	0.8	0.57	0.3	0.6	0.40
Myotis nattereri	0.75	1	0.33	0.48	0.39	0.43	0.76	0.55
Nyctalus lasiopterus	/	/	0	0	0	/	/	/
Nyctalus leisleri	0.55	0.81	0.47	0.63	0.54	0.38	0.85	0.53
Nyctalus noctula	0.55	0.72	0.55	0.84	0.67	0.30	0.84	0.44
Pipistrellus kuhlii	1	/	0.54	0.74	0.62	/	/	/
Pipistrellus nathusii	0.5	1	0.24	0.39	0.30	0.28	0.85	0.43
Pipistrellus pipistrellus	0.67	0.92	0.44	0.6	0.51	0.36	0.76	0.48
Pipistrellus pygmaeus	0.12	0.06	0.48	0.83	0.61	0.3	0.98	0.46
Rhinolophus ferrumequinum	/	/	0.38	1	0.55	0.57	1	0.73
Rhinolophus hipposideros	/	/	0.67	0.73	0.70	0.71	0.91	0.80
Vespertilio murinus	/	/	0.29	0.44	0.35	/	/	/
Plecotus auritus	0.33	0.33	0.39	0.63	0.48	0.40	0.83	0.54
Plecotus austriacus	0	0.33	0	0	0	0.42	1	0.59
Global	**0.47**	**0.69**	**0.39**	**0.57**	**0.48**	**0.42**	**0.78**	**0.54**

Table 5.3: Performance of BattyBirdNET and BatDetect2 for species prediction. '/' indicates that no data was available.

It can be noticed in Table 5.3, looking at the last row, that BatDetect2 is better than BattyBirdNET.

Indeed, the mean *recall* of BatDetect2 is higher than the mean *recall* of BattyBirdNET, in both datasets. In Xeno-Canto dataset, the mean *precision* is also higher, and thus the *F1-score* too.

One can also observe that for each species, the *recall* is always higher than the *precision*, as we wanted in this context of bats call identification.

For Lauzelle dataset, *empty*, *Eptesicus serotinus*, *Myotis myotis*, *Pipistrellus pipistrellus* are the classes the most frequent in the dataset, so the performance measures of every other class are less reliable and must be taken carefully.
About Xeno-Canto dataset, which is a little bit more complete than the first one, *empty*, *Barbastella barbastellus*, *Eptesicus nilssonii*, *Nyctalus leisleri*, *Nyctalus noctula*, *Pipistrellus pipistrellus*, *Pipistrellus pygmaeus* are the classes the most frequent in the dataset, so the performance measures of every other class are less reliable and must be taken carefully.

Bats species groups

		Lauzelle	Xeno-Canto		
		Recall	Precision	Recall	F1-score
empty	BatML	0.03	0.26	0.41	0.32
	BattyBirdNET	0.625	0.86	0.9	0.88
	BatDetect2	0.94	0.81	0.81	0.81
Barbarg	BatML	/	0.18	0.67	0.28
	BattyBirdNET	/	0.75	0.93	0.83
	BatDetect2	/	0.58	0.96	0.73
Envsp	BatML	0.98	0.32	0.82	0.46
	BattyBirdNET	0.94	0.74	0.85	0.79
	BatDetect2	0.98	0.60	0.94	0.73
Myosp	BatML	0.98	0.41	0.93	0.56
	BattyBirdNET	0.77	0.58	0.82	0.68
	BatDetect2	0.99	0.55	0.90	0.68
Pip35	BatML	1	0.3	0.85	0.44
	BattyBirdNET	0.94	0.56	0.71	0.63
	BatDetect2	1	0.36	0.85	0.51
Pip50	BatML	0.98	0.3	0.91	0.44
	BattyBirdNET	0.67	0.49	0.70	0.57
	BatDetect2	0.92	0.43	0.87	0.58
Plesp	BatML	0.71	0.28	0.65	0.39
	BattyBirdNET	0.57	0.43	0.67	0.52
	BatDetect2	0.71	0.52	0.88	0.66
Rhisp	BatML	/	0.31	0.89	0.46
	BattyBirdNET	/	0.48	0.84	0.62
	BatDetect2	/	0.69	0.95	0.80
Global	**BatML**	**0.78**	**0.3**	**0.76**	**0.43**
	BattyBirdNET	**0.75**	**0.61**	**0.8**	**0.69**
	BatDetect2	**0.92**	**0.57**	**0.9**	**0.70**

Table 5.4: Performance of BatML, BattyBirdNET and BatDetect2 for species groups prediction.
'/' indicates that no data was available.

It can be noticed in Table 5.4, in the last row, that BatDetect2 is better than the two others, in the case of Lauzelle dataset.
In the case of Xeno-Canto dataset, BatML is clearly less efficient than the two others. Its main drawback is its *precision* score which has a very negative impact on its *F1-score*. BatML tends to predict a lot of different species groups and makes a lot of wrong predictions. It is also very bad at identifying files without any bats, as shown in the first row.

When looking at the mean *F1-score* of BattyBirdNET and BatDetect2, they are quite similar. The difference lays in *precision-recall* trade-off. BattyBirdNET has a slightly higher *precision*, while BatDetect2 has a better *recall*. Because of the context of bats call identification, it is better to have a higher *recall*, thus one can conclude that BatDetect2 is still the best of all 3 AIs.

5.3.2 Results with other configurations

In this part, the way to construct the confusion matrix is changed. Instead of taking into account all predicted species/groups by the AI, only **one** species/group is predicted, which is the species/group with the most occurrences in a given file. Therefore, a file is correctly classified if the most frequent predicted class is the same as the class associated to this file. In this case, the element `(true class, true class)` in the confusion matrix is still incremented by one. For the confusion matrix with the Xeno-Canto dataset, no matter the predicted class, the element `(true class, predicted class)` is incremented by one, increasing thus the diagonal element if the predicted class is the right one.

This is an interesting way to test the 3 AIs because one might be interested in retrieving only **one** species for a whole audio, and not necessarily in all detected calls potentially associated to different species.

Doing so obviously results in lower *recall* because it's no longer enough to predict the right species at least once; the right species has to be the most predicted one. However, it might end up giving a higher *precision* since FP is no longer increased for each wrong predicted species, given that only **one** species is predicted.

Bats species

Species	Lauzelle		Xeno-Canto					
	BattyBirdNET	BatDetect2	BattyBirdNET			BatDetect2		
	recall	recall	precision	recall	F1-score	precision	recall	F1-score
nothing	0.63	0.94	0.89	0.90	0.90	0.81	0.81	0.81
Barbastella barbastellus	/	/	0.93	0.93	0.93	0.94	0.94	0.94
Eptesicus nilssonii	/	/	0.56	0.61	0.58	/	/	/
Eptesicus serotinus	0.54	0.62	0.73	0.74	0.74	0.85	0.85	0.85
Myotis alcathoe	/	/	0	0	/	1.00	1.00	1.00
Myotis bechsteinii	/	/	0.17	0.20	0.18	0.40	0.40	0.40
Myotis brandtii	0	0	0	0	/	0.25	0.25	0.25
Myotis capaccinii	/	/	0	0	0	/	/	/
Myotis dasycneme	/	/	0.80	0.80	0.80	/	/	/
Myotis daubentonii	0.37	1.00	0.43	0.53	0.47	0.76	0.76	0.76
Myotis emarginatus	0	0	0.75	1.00	0.86	/	/	/
Myotis myotis	0	0	0.65	0.67	0.69	/	/	/
Myotis mystacinus	0.29	0.50	0.6	0.60	0.60	0.40	0.40	0.40
Myotis nattereri	0.50	0.75	0.45	0.48	0.47	0.59	0.62	0.60
Nyctalus lasiopterus	/	/	0	0	0	/	/	/
Nyctalus leisleri	0.33	0.33	0.54	0.59	0.56	0.62	0.64	0.63
Nyctalus noctula	0.45	0.38	0.76	0.77	0.76	0.62	0.63	0.63
Pipistrellus kuhlii	1.00	0	0.63	0.67	0.65	/	/	/
Pipistrellus nathusii	0.13	0.63	0.27	0.30	0.29	0.67	0.67	0.67
Pipistrellus pipistrellus	0.50	0.79	0.49	0.53	0.51	0.72	0.74	0.73
Pipistrellus pygmaeus	0	0	0.70	0.77	0.73	0.91	0.92	0.92
Rhinolophus ferrumequinum	/	/	0.73	1.00	0.84	1.00	1.00	1.00
Rhinolophus hipposideros	/	/	0.73	0.73	0.73	0.91	0.91	0.91
Vespertilio murinus	/	/	0.32	0.35	0.34	/	/	/
Plecotus auritus	0.33	0.33	0.51	0.57	0.54	0.57	0.59	0.58
Plecotus austriacus	0	0	0	0	0	0.60	0.60	0.60
Global	**0.32**	**0.39**	**0.49**	**0.52**	**0.50**	**0.70**	**0.71**	**0.70**
Global	**0.47**	**0.69**	**0.39**	**0.57**	**0.48**	**0.42**	**0.78**	**0.54**

Table 5.5: Performance of BattyBirdNET and BatDetect2 with the most frequent species being predicted. The last row is the row coming from Table 5.3 to allow a better comparison.
'/' indicates that no data was available.

From Table 5.5, it can be noticed the *recall* is always lower, as predicted earlier, with this new configuration. The difference is big with the Lauzelle dataset, but rather small concerning the Xeno-Canto dataset. As predicted too, the *precision* is better.

Even with this new configuration, BatDetect2 remains better than BattyBirdNET in both *precision* and *recall* scores.

Bats species groups

		Lauzelle	Xeno-Canto		
		Recall	Precision	Recall	F1-score
empty	BatML	0.3	0.41	0.41	0.41
	BattyBirdNET	0.63	0.89	0.90	0.89
	BatDetect2	0.94	0.81	0.81	0.81
Barbarg	BatML	/	0.38	0.42	0.40
	BattyBirdNET	/	0.92	0.92	0.92
	BatDetect2	/	0.92	0.92	0.92
Envsp	BatML	0.71	0.61	0.61	0.61
	BattyBirdNET	0.74	0.83	0.83	0.83
	BatDetect2	0.79	0.84	0.84	0.84
Myosp	BatML	0.83	0.84	0.84	0.84
	BattyBirdNET	0.39	0.75	0.78	0.76
	BatDetect2	0.83	0.82	0.82	0.82
Pip35	BatML	0.81	0.60	0.61	0.61
	BattyBirdNET	0.94	0.61	0.63	0.62
	BatDetect2	0.63	0.62	0.62	0.62
Pip50	BatML	0.78	0.76	0.76	0.76
	BattyBirdNET	0.49	0.60	0.64	0.62
	BatDetect2	0.79	0.81	0.83	0.82
Plesp	BatML	0.67	0.29	0.29	0.29
	BattyBirdNET	0.50	0.57	0.59	0.58
	BatDetect2	0.67	0.63	0.65	0.64
Rhisp	BatML	/	0.75	0.79	0.77
	BattyBirdNET	/	0.73	0.84	0.78
	BatDetect2	/	0.95	0.95	0.95
Global	**BatML**	**0.68**	**0.58**	**0.59**	**0.58**
	BattyBirdNET	**0.61**	**0.74**	**0.77**	**0.75**
	BatDetect2	**0.77**	**0.80**	**0.80**	**0.80**
Global	**BatML**	**0.78**	**0.3**	**0.76**	**0.43**
	BattyBirdNET	**0.75**	**0.61**	**0.8**	**0.69**
	BatDetect2	**0.92**	**0.57**	**0.9**	**0.70**

Table 5.6: Performance of BatML, BattyBirdNET and BatDetect2 with the most frequent species group being predicted. The last row is the row coming from Table 5.4 to allow a better comparison.
'/' indicates that no data was available.

Table 5.6 confirms the fact the BatDetect2 is the most efficient, while BatML is the less efficient.

5.4 Conclusion

This chapter has tried to evaluate the performance of BatML, BattyBirdNET and BatDetect2. As a reminder, we had to assess the performance by performing multi-class classification, rather than multi-label classification, which is initially the purpose of these 3 AIs'. This is because we had no available data suitable for multi-label classification, which is much harder and more time consuming for experts to collect such data.

The experiments have shown that BatDetect2 is the most reliable model, while BatML is the less efficient one.

Chapter 6

Creation of an ensemble model

After completing the application, we wanted to work toward enhancing one of our AIs, we decided to create an ensemble model using multiple of our already present AIs, ensemble models are combinations of models that aim to use the best parts of each AI that composes them to get a better or more stable result.

6.1 Techniques

Ensemble models use several learning algorithms in order to obtain better results than those obtained by any of the single algorithm.

Here are some techniques we explored for our ensemble model:

6.1.1 Bagging

Bagging [10] consists in training multiple similar models with different parts of a dataset, then taking an average or majority of the results to output a prediction.

This algorithm begins by bootstrapping, which is a sampling technique used to obtain different data samples by picking random subsets of a dataset. Each learner is then trained with one of the subsets. Usually, each model uses the same learning algorithm.

Ultimately, as mentioned previously, the result is generated by taking the average or the majority from the learners.

6.1.2 Stacking

As opposed to bagging which trains similar models with different data, stacking [36] trains different models with the same data, to explore a problem from different angles.
A new model is then trained on the predictions of the base models

6.1.3 Weighted Voting

The last technique we considered is weighted voting, it consists in taking a pool of base algorithms and make them perform some task, then comparing their results to make a final decision based on the combined weighted outcomes and probabilities of each algorithm.

It is proved [24] that the upper bound on the prediction mistakes made by a pool of algorithms is:

$$O(\log |A| + m)$$

With A being the sequence of predictions and m being the most amount of mistakes made by the best algorithm from the pool. In other words, the upper bound scales with the best performing algorithm in the pool for the task.

We ended up choosing weighted voting over the other techniques, as this technique is straightforward to implement and does not require the additional training of multiple models or a meta-learner.
This allows to reduce computational overhead and complexity, making it a practical choice. Additionally, this method also provides flexibility, allowing us to dynamically adjust the weights based on the performance of each model, which is beneficial for handling diverse data and varying conditions.

6.2 Choosing the AIs

In this section, we are quickly going to go over which AI we chose for the ensemble and how they function.

6.2.1 BatML

BatML classifies audio files by performing multi-label predictions, in other words, instead of giving a prediction on the whole file, it will return a classification for each detected call.

BatML was created with multiple trained models, we chose to use the one called 'cnn2' as it was the most efficient for the time needed for classification.

This model trains two CNN neural networks, one for detection and the other for classification. The detection CNN has two labels, either the timestamp has a bat call or it has not. The second CNN classifies between 7 labels, these are the bat families that were present in the dataset used for training that was provided by Natagora, namely:

Family	Name
1	Barbastella (Barbarg)
2.1	Epistecus (Envsp)
2.2	Nyctalus (Envsp)
2.3	Vespertilio (Envsp)
3	Myotis (Myosp)
4	Pipistrellus (Pip35)
5	Pipistrellus (Pip50)
6	Plecotus (Plesp)
7	Rhinolophus (Rhisp)

Table 6.1: List of bat families used for classification by BatML.

The algorithm starts by computing features from the audio file, then passes them through the detection CNN to get the timestamps that contain bat calls, the timestamps are then given to the classification CNN to predict the family of the detected bat.
The AI returns the detections that are above a given probability threshold.

Here is a prediction of BatML on a file containing bats from the Nyctalus family, visualised by EchoExplorer:

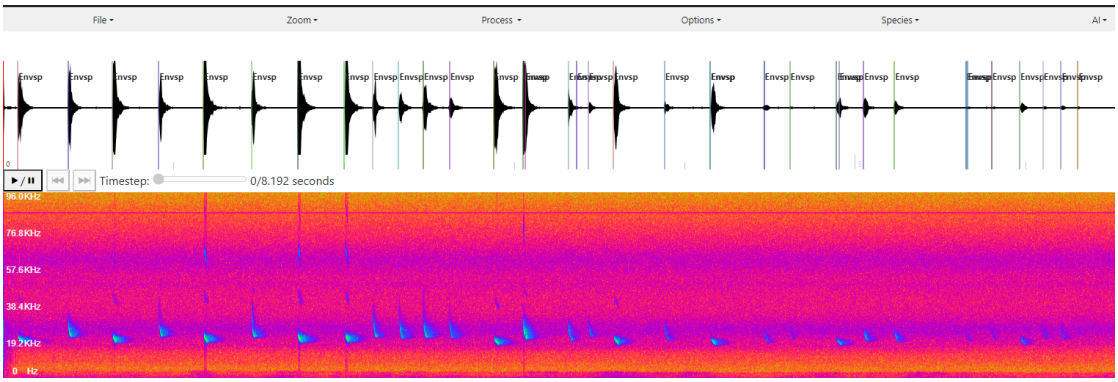


Figure 6.1: Visualisation of BatML's results on an audio file containing Nyctalus noctula in EchoExplorer.

6.2.2 BatDetect2

Similarly to BatML, BatDetect2 performs multi-label detection on audio files.

To achieve this, BatDetect2 uses the audio detector of another model, Tadarida [5], that detects acoustic events before comparing the spectrograms of these events with ground truths using NMS (non-maximum suppression) to label each to a bat specie or discard it.

The model classifies audio events between 17 bat species that were present in the 5 datasets that were presented in the chapter on AIs **??**

Here is a prediction of BatDetect2 on a file containing bats from the Nyctalus family, visualised by EchoExplorer:

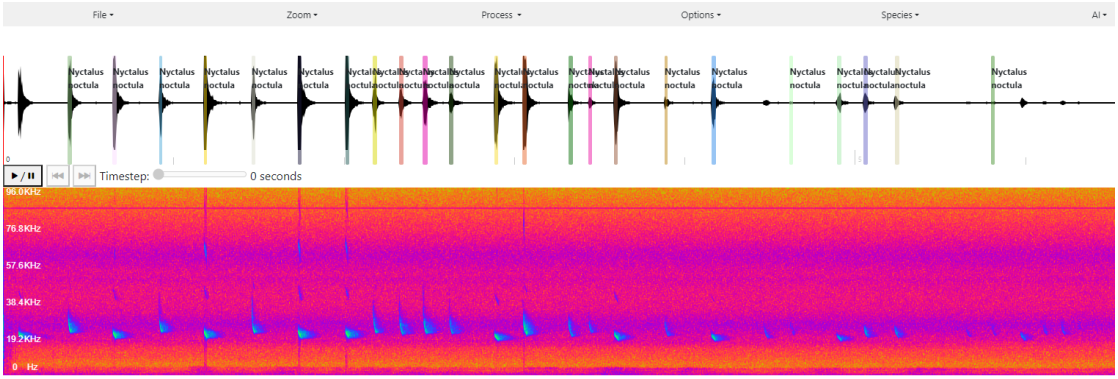


Figure 6.2: Visualisation of BatDetect2's results on an audio file containing Nyctalus noctula in EchoExplorer.

6.3 Implementation

The first step to implement the weighted voting was to modify the AIs to return all detected calls, including those with low probabilities, along with the probabilities for each label.

Filename	**Timestamp**	**Prob. of being a bat**	**Prob. of belonging to each Label**

Table 6.2: new CSV header returned by BatML and BatDetect2.

Next, for each AI, we pre-process the outputs to remove calls that are too close to each other according to a time window, as they can be considered the same event.

We normalise the probabilities to sum to 1.

We then begin to merge the outputs of the two AIs by finding pairs of calls between their results. We noticed that the timestamps from BatML are usually slightly earlier than those from BatDetect2. Therefore, we decided to disregard the fact that BatDetect2 returned timeframes and only keep the start of the calls.

After creating a new dataframe with the paired calls using the Pandas library, we address the biggest difference between the AIs: BatDetect2 classifies in more specific labels, while BatML only classifies in families.
Fortunately each species identified by BatDetect2 can be mapped to a family from BatML. This prompted us to add a new list to the dataframe with the mapped probabilities of BatDetect2 to their respective families.

Lastly, the voting is performed. Before explaining the algorithm, here is a list of the parameters that the ensemble takes:

Parameter	Description
`timewindow`	The time-window under which two calls are classified as 'close'
`batml_weight`	The weight given to the predictions of BatML
`batdetect_weight`	The weight given to the predictions of BatDetect2
`batml_minbatprob`	The minimum threshold for the probability of being a bat call from BatML
`batdetect_minbatprob`	The same threshold for BatDetect2
`merged_minbatprob`	The threshold for the weighted sum of the probabilities
`groupprob_threshold`	The minimum threshold for the probability of a call belonging to a family
`specprob_threshold`	The minimum threshold for the probability of a call belonging to a specific species

Table 6.3: List of parameters for the ensemble algorithm.

The weights were assigned according to the results of each AI, since BatDetect2 has better results, and is also usually more reserved on its predictions, assigning it a higher weight helps the model.

Here is how the algorithm works:

- If the call is predicted only by BatML, it is considered if the probability of being a bat call is above the threshold.

The call is kept and classified as the label with the highest confidence level, provided this level is above the threshold.

- If the call is predicted only by BatDetect2, the process is similar, except we first check if the probability of being a specific specie is above the threshold before checking for the family.
- If the call is present in both detections, we check the weighted sum of the probabilities, then the weighted sum for the families, and finally the probability for the species from BatDetect2.

The detections are then returned in this format:

FileName	Timestamp	Label	Confidence Level

Table 6.4: new CSV header returned by the ensemble model.

6.4 Evaluation

After adding the ensemble model to EchoExplorer, users can now call the model on their files.

Here is an example of the model on the previous file containing bats from the Nyctalus family, visualised by EchoExplorer:

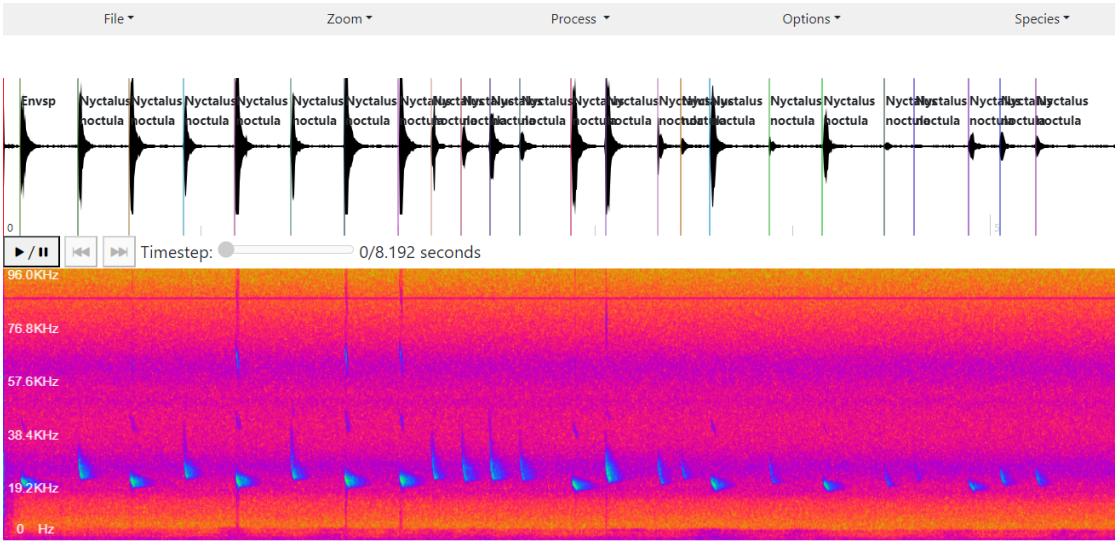


Figure 6.3: Visualisation of the ensemble model's results on an audio file containing Nyctalus noctula in EchoExplorer.

We also ran similar tests as in Chapter 5 comparing the ensemble model to batdetec2:

Species	Lauzelle		Xeno-Canto					
	AIVoting	BatDetect2	AIVoting			BatDetect2		
	recall	recall	precision	recall	F1-score	precision	recall	F1-score
empty	0.13	0.94	0.04	0.08	0.05	0.81	0.81	0.81
Barbarg	/	/	0.57	0.96	0.71	0.58	0.96	0.73
Envsp	0.98	0.98	0.55	0.96	0.70	0.60	0.94	0.73
Myosp	0.98	0.99	0.57	0.91	0.70	0.55	0.90	0.68
Pip35	1	1	0.42	0.8	0.55	0.36	0.85	0.51
Pip50	0.52	0.92	0.17	0.3	0.22	0.43	0.87	0.58
Plesp	0.83	0.71	0.48	0.88	0.63	0.52	0.88	0.66
Rhisp	/	/	0.54	0.95	0.69	0.69	0.95	0.80
Global	0.74	0.92	0.42	0.73	0.53	0.57	0.90	0.70

Table 6.5: Performance of AIVoting compared to BatDetect2.
'/' indicates that no data was available.

Species	Lauzelle		Xeno-Canto					
	AIVoting	BatDetect2	AIVoting			BatDetect2		
	recall	recall	precision	recall	F1-score	precision	recall	F1-score
empty	0.13	0.94	0.08	0.08	0.08	0.81	0.81	0.81
Barbarg	/	/	0.85	0.89	0.87	0.92	0.92	0.92
Envsp	0.92	0.79	0.90	0.93	0.91	0.84	0.84	0.84
Myosp	0.93	0.83	0.87	0.90	0.88	0.82	0.82	0.82
Pip35	1	0.63	0.71	0.73	0.72	0.62	0.62	0.62
Pip50	0.3	0.79	0.18	0.20	0.19	0.81	0.83	0.82
Plesp	0.67	0.67	0.74	0.74	0.74	0.63	0.65	0.64
Rhisp	/	/	0.95	0.95	0.95	0.95	0.95	0.95
Global	0.66	0.77	0.66	0.68	0.67	0.80	0.80	0.80

Table 6.6: Performance of AIVoting compared to BatDetect2, with the most frequent species being predicted.
'/' indicates that no data was available.

We can clearly observe that BatDetect2 outperforms the weighted voting. This is not surprising as the parameters were not yet formally trained, further improvements to the ensemble model are discussed in the next Chapter.

Chapter 7

Further improvements

This chapter aims to present improvements that we believe can be made to the web application, but we did not have time to implement.

7.1 Scalability

One key area for further improvement in our Flask web application is its scalability. As the number of users and the volume of audio files to process grows, ensuring the application can handle increased load efficiently is critical.

7.1.1 Empirical analysis

This section presents an analysis of the CPU/GPU and memory usage of the second Flask application in charge of processing audio files, under different levels of simulated user load. The aim is to understand how the system resources are utilised when multiple users simultaneously upload files for processing. Then, this would help us to better tackle horizontal scaling, and to better estimate the number of new servers to be added so that the application can handle x users at the same time.

To analyse the CPU and memory usage, we used the command line `top`. This command provides a real-time view of the processes running and their resource usage. To analyse the GPU memory usage, we used the command line `nvidia-smi`, providing also a real-time view of the processes running and their memory usage.

```
top - 21:15:54 up 45 days, 12:08,  1 user,  load average: 0,24, 0,08, 0,02
Tasks: 299 total,   3 running, 296 sleeping,   0 stopped,   0 zombie
%Cpu(s): 12,4 us,  0,1 sy,  0,0 ni, 87,4 id,  0,0 wa,  0,0 hi,  0,0 si,  0,0 st
MiB Mem :  31658,8 total,  21161,5 free,   5982,8 used,   4514,5 buff/cache
MiB Swap:  15996,0 total,  15643,1 free,    352,9 used.  25093,0 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
 864169 batmen    20   0   12,9g   3,2g 659572 R  99,7  10,2   0:09.47 python3
 864133 batmen    20   0   12,9g   3,2g 658944 R  99,3  10,2   0:10.22 python3
 860524 batmen    20   0 1085976  70864  12320 S   0,7   0,2   1:53.99 python3
 863903 batmen    20   0   65560   5040   4156 R   0,3   0,0   0:00.47 top
      1 root      20   0  238732   7484   4856 S   0,0   0,0   3:27.97 systemd
      2 root      20   0       0      0      0 S   0,0   0,0   0:02.01 kthreadd
      3 root       0 -20       0      0      0 I   0,0   0,0   0:00.00 rcu_gp
      4 root       0 -20       0      0      0 I   0,0   0,0   0:00.00 rcu_par_gp
      6 root       0 -20       0      0      0 I   0,0   0,0   0:00.00 kworker/0:0H-even+
      9 root       0 -20       0      0      0 I   0,0   0,0   0:00.00 mm_percpu_wq
     10 root      20   0       0      0      0 S   0,0   0,0   0:00.72 ksoftirqd/0
     11 root      20   0       0      0      0 I   0,0   0,0   7:08.42 rcu_sched
     12 root      rt   0       0      0      0 S   0,0   0,0   0:00.63 migration/0
     13 root      rt   0       0      0      0 S   0,0   0,0   0:00.00 watchdog/0
     14 root      20   0       0      0      0 S   0,0   0,0   0:00.00 cpuhp/0
     15 root      20   0       0      0      0 S   0,0   0,0   0:00.00 cpuhp/1
     16 root      rt   0       0      0      0 S   0,0   0,0   0:03.41 watchdog/1
     17 root      rt   0       0      0      0 S   0,0   0,0   0:00.74 migration/1
     18 root      20   0       0      0      0 S   0,0   0,0   0:00.30 ksoftirqd/1
```

Figure 7.1: `top` example with 2 users using BatDetect2.

On Figure 7.1, one can see that the 2 first processes are the ones dedicated to the 2 users using BatDetect2.
'%MEM' column indicates the proportion of the total memory used by the process. While '%CPU' indicates the proportion of one logical processor's capacity used by the process.
The Gotham machine has 8 physical cores, and with hyper-threading, each physical core can handle two threads simultaneously. This means, that in total, 16 tasks can be run simultaneously, this is also the number of logical processors.
In the top left corner '%Cpu(s)' indicates the total proportion of CPU's capacity used. Thus, if we sum up each element of '%CPU' column and divide it by 1600% (because 16 logical processors), we should get the value 12.4 indicated. In this case:

$$\frac{99.7 + 99.3 + 0.7 + 0.3}{1600} = 12.5$$

Methodology

To obtain the results presented in the next section, we manually uploaded files of 5 minutes, except for rows with 20 files, which are 30 seconds files. Unfortunately, we were unable to simulate more than 6 users at the same time, because we could only simulate one user per browser.

Results

About the GPU usage, only the code of BatML is adapted to use it. For one user, whatever the size and number of audio files, it uses 23% of its total memory. This means that BatML can serve, using its GPU, 4 users simultaneously.

nUsers-nFiles	CPU [%]				MEM [%]			
	BatML	BirdNET	BattyBirdNET	BatDetect2	BatML	BirdNET	BattyBirdNET	BatDetect2
1-1	6.9	6.3	6.3	6.3	8.5	1.5	2	11
1-3	6.9	18.7	18.7	6.3	9	4.5	6	11
1-5	6.9	31.2	31.2	6.3	9	7.5	10	11
1-20	6.9	50	50	6.3	8.5	9.6	10.4	10.2
2-1	13.5	12.5	12.5	12.5	16	3	4	22
2-3	13.5	37.5	37.5	12.5	18	9	12	22
2-5	13.5	50	50	12.5	18	12	16	22
2-20	13.5	50	50	12.5	17	9.6	10.4	20.4
6-5	40.5	74.2	80.9	37.5	51	15.6	16.9	61.2
6-20	40.5	68.7	74.1	37.5	51	13.2	15.5	61.2

Table 7.1: Proportion of total resource utilisation.
The first column indicates the number of simultaneous users and number of files uploaded per user. Note that 6.3% of CPU corresponds to one process using 100% of CPU's capacity of one logical processor.

From Table 7.1, one can observe that BatML also uses the CPU. It uses more than 100% ($0.069 \times 16 = 110\%$), which might be due to the fact that the process uses 2 threads thanks to hyper-threading.
It can be noticed that BirdNET and BattyBirdNET uses more processes if the number of files increases. This is because they both work with a threads pool to accelerate the analysis, instead of doing it sequentially.

- BatML uses one process per user, requiring 6.9% of the total CPU capacity and 9% of the total memory. Theoretically, this means that BatML can serve 11 users at the same time, limited due to the memory capacity.
- BatDetect2 uses one process per user, requiring 6.3% of the total CPU capacity and 11% of the total memory. In theory, this means that BatDetect2 can serve 11 users at the same time, limited due to the memory capacity.
- BirdNET and BattyBirdNET both use more processes if any are available to speed up the analysis when multiple files are uploaded. They use small amount of memory and are therefore limited to 15 users at a time due to CPU capacity.

In conclusion, with this empirical analysis, we conclude that the application can serve a maximum of 11 users at the same time, limited by BatML and BatDetect2.

7.1.2 Horizontal scaling

The big bottleneck of the application is the processing part of the audio files by the AIs.
For the moment, the Flask application on Gotham that handles the audios processing is able to process only a few requests at the same time. The number of simultaneous requests it is able to process is limited by the Gotham's resources. To be able to process more requests at the same time without modifying the complexity of the AIs, we need to have other instances of the Flask application to deal with the files processing.
We would therefore need one or several additional machines like Gotham to host these multiple instances that would run simultaneously. This is horizontal scaling.

Here are key steps to implement horizontal scaling:

1. Use Docker [26] to containerise the application to make it easier to replicate and deploy multiple instances. Use Docker Compose to handle the multiple instances.
2. Select a load balancer to distribute incoming requests across multiple instances of the application.
3. (Use Kubernetes [11], which is a containers orchestrator, that is able to dynamically adjust the number of running instances based on resource utilisation and demand, ensuring optimal performance).

7.1.3 Requests management

Horizontal scaling of the main Flask application is not necessarily relevant because it does not really have a bottleneck and all operations done by this Flask application are really quick, compared to the processing task made on Gotham.

However, a great improvement to bring to the main Flask application would be to use asynchronous requests for the audio processing tasks.
Indeed, it would be better to implement an asynchronous task queue because these are much more suitable to handle long-time processing tasks. With synchronous tasks, the process that is handling a request will block on the call that executes the long-time processing task. However, with asynchronous task, the process can be paused on the blocking call and still processes other incoming requests until the blocking call has finished.

For now, the file processing routine is made in a synchronous way, with the main Flask sending a request to the Gotham application with the files to process, then being blocked until it receives the response from Gotham.
Each time a user makes a request to analyse a file, it is sent to the Gotham application. But if the maximum number of simultaneous requests being processed is reached, the new incoming request end up causing an error (i.e. the response of Gotham is empty) because there is no system, such as a tasks queue, to manage all the incoming requests.
Knowing that, we believe it would be necessary to implement an asynchronous tasks queue on the Gotham Flask application and all other duplicate instances from horizontal scaling.

In practice, Celery [34] is a common solution to implement an asynchronous tasks processing queue in a Flask application.

7.1.4 Databases

In the web application, we have two databases. The first one uses SQLAlchemy, and the second one uses PyMongo. Given that the horizontal scaling is much more interesting to apply to the second Flask application on Gotham, there is no need to adapt the two databases because they only interact with the main Flask application.

7.2 AI explainability

Before creating the ensemble model, we wanted to make a choice between two options; The first one was to make the ensemble model. The second one was to try to make one of the AIs explainable or create a new explainable model, which is a process that aims to reduce the 'black box' side of an AI by making it return information on its decision-making in some way, and thus being more comprehensible by humans.

Having an explainable model would be very interesting and useful, as explaining why some results are returned could help either the experts validate or infirm some prediction, or it could help amateurs understand and learn interesting features of bat detection. Linking this with an improved and more interactive spectrogram on the front-end of the application could really enhance the usefulness of the AI.

We could imagine the AI returning the coordinates of the predicted call, then the spectrogram making clickable spaces that would contain the most meaningful features for classification.

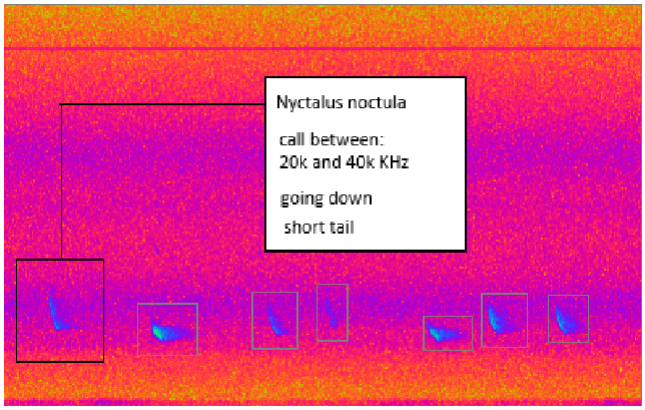


Figure 7.2: Speculative view of an improved spectrogram of EchoExplorer using explainable AI.

Ultimately, we ended up going in the direction of the ensemble model, as creating a whole new model seemed like a tall task, given the remaining time and the fact that we would be walking in the dark given our limited previous experiences in the field.
However, we deemed important to mention this option as it seems to have great potential for the future.

7.3 Ensemble Model

Since creating the ensemble model was the last implementation process we tackled, there are several areas that could be further explored to enhance its performance.

7.3.1 Parameter training

Currently, the parameters we use for the classification computation are set manually, based on our best estimates.

We could use part of our testing dataset to train these parameters, optimising them to achieve the best possible results.

7.3.2 Adding a third AI

The model currently uses two AIs; however, we have a third AI, BattyBirdNET, that we use in our application.

BattyBirdNET splits audios into time windows of a few seconds and attempts to classify species within each window before returning a single label for each.

Here is a prediction of BattyBirdNET on a file containing bats from the Nyctalus family, visualised by EchoExplorer:

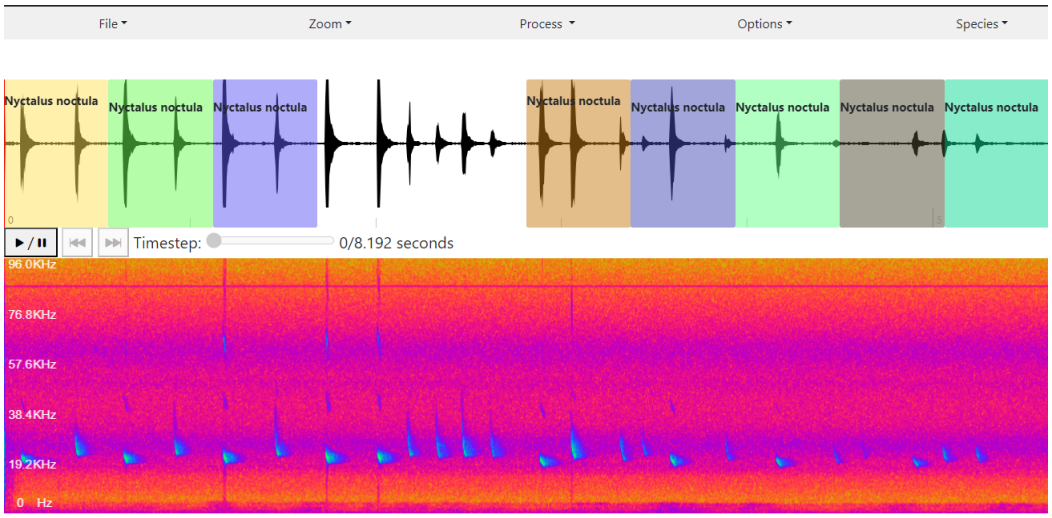


Figure 7.3: Visualisation of BattyBirdNET's results on an audio file containing noctules in EchoExplorer.

Adding BattyBirdNET to the ensemble would require some adaptations, considering it is not inherently multi-label. Its weight could be reduced compared to the other AIs due to the less precise time window. However, having more predictions by incorporating another AI is often beneficial in ensemble models.

7.3.3 Data Usage

One of the first objectives of creating EchoExplorer was to gather usable validated data, which could then be used to train the models we use in the application.

We have already implemented the data validation process with the expert role, but as of now, the data is not fully integrated into a training pipeline.

An important improvement could be to automate the integration of validated data into the model training process. This would involve creating workflows that automatically update the training datasets with our newly validated data, then retrain the models periodically and deploy the updated models.

This would ensure that the models continuously improve with the addition of new expert-validated data, providing more accurate and reliable call detection and identification.

Chapter 8

Conclusion

This thesis was aimed at building a web application that facilitates the use of automated animal call identification tools. As previously mentioned, most of the available open-source tools do not include a graphical user interface, making them more difficult to use for people with no computer science skills.

EchoExplorer focuses on models made for bat species. We integrated four AI models, three of which are dedicated to bat species; (i) BatML, (ii) BattyBirdNET, (iii) BatDetect2. The last one, BirdNET, is specialised in bird species. The application successfully integrates these four tools and is constructed to be flexible with the objective of easing the integration of any new AI.

The application is not only capable of processing audio recordings with AIs but also provides features to help users manually analyse and visualise their audio recordings. EchoExplorer displays a spectrogram as well as the waveform of the audio file. Moreover, an interactive table summarises the list of identified calls, and filtering options are available to allow searching only for specific species, comparing the results of different AIs, and removing less likely calls.

The web application allows users to save or load their own annotations, as well as those made by the AIs. Users have a personal space in which they can filter their audios by species, making it easier to find and check the presence of particular species in many analysed files.

Finally, we have implemented a system to enable the validation of annotations by experts. Users can share their recordings analysed by the AIs, after which, in a similar way that user can see and filter their files, experts can search for files containing specific species to validate.

What motivated us to implement this system was that we wanted to enable users to be part of a community and share their recordings or receive help from others to confirm and validate their annotations, in a similar fashion to *observation.org*. We also wanted to gather a collection of labelled data of bat calls, as gathering labelled data is often a difficult task in machine learning, as well as an essential step for the creation of an AI model. By collecting labelled data, we would be able to further train the AI models, enabling them to perform better, for the benefit of users.

In conclusion, EchoExplorer represents a significant step toward making animal call identification tools accessible to a broader audience. By creating a community of users and experts, we not only enhance the accuracy and reliability of species identification but can also contribute to scientific research as well as wildlife monitoring and conservation.

Bibliography

[1] Volodymyr Agafonkin et al. *Leaflet*. 2012. URL: https://leafletjs.com/.

[2] Amazon Web Services. *boto3: Amazon Web Services SDK for Python*. 2015. URL: https://boto3.amazonaws.com/v1/documentation/api/latest/index.html.

[3] Oisin Mac Aodha et al. "Towards a General Approach for Bat Echolocation Detection and Classification". In: *bioRxiv* (2022). DOI: 10.1101/2022.12.14.520490. eprint: https://www.biorxiv.org/content/early/2022/12/16/2022.12.14.520490.full.pdf. URL: https://www.biorxiv.org/content/early/2022/12/16/2022.12.14.520490.

[4] Michel Barataud. *Ecologie acoustique des Chiroptères d'Europe, identification des espèces, étude de leur habitats et comportements de chasse*. 3rd. Biotope Editions, 2015. ISBN: 978-2-36662-142-6.

[5] Yves Bas, Didier Bas, and Jean-François Julien. "Tadarida: A Toolbox for Animal Detection on Acoustic Recordings". In: *Journal of Open Research Software* 5.1 (Jan. 2017), p. 6. DOI: 10.5334/jors.154. URL: https://hal.science/hal-04084353.

[6] Michael Bayer. "SQLAlchemy". In: *The Architecture of Open Source Applications Volume II: Structure, Scale, and a Few More Fearless Hacks*. Ed. by Amy Brown and Greg Wilson. aosabook.org, 2012. URL: http://aosabook.org/en/sqlalchemy.html.

[7] Mélanie Beauvois and Lucile Dierckx. "Automated detection of bat species in Belgium". MA thesis. Ecole polytechnique de Louvain, Université catholique de Louvain, 2021. URL: http://hdl.handle.net/2078.1/thesis:30594.

[8] Biotope. *Biotope, l'entreprise de l'écologie*. URL: https://www.biotope.fr/.

[9] Biotope. *Sonochiro®, le logiciel d'analyse automatique d'ultrasons de chauves-souris*. 2017. URL: https://sonochiro.biotope.fr/.

[10] L Breiman. "Bagging predictors". In: *Machine Learning* (1996).

[11] Eric A. Brewer. "Kubernetes and the path to cloud native". In: *Proceedings of the Sixth ACM Symposium on Cloud Computing*. SoCC '15. Kohala Coast, Hawaii: Association for Computing Machinery, 2015, p. 167. ISBN: 9781450336512. DOI: 10.1145/2806777.2809955. URL: https://doi.org/10.1145/2806777.2809955.

[12] Chris Scott. *Batclassify*. URL: https://bitbucket.org/chrisscott/batclassify/src/master/.

[13] Max Countryman. *Flask-Bcrypt*. 2011. URL: https://flask-bcrypt.readthedocs.io/en/1.0.1/index.html.

[14] *DataTables*. URL: https://datatables.net/.

[15] *Flask-Limiter*. 2014. URL: https://flask-limiter.readthedocs.io/en/stable/.

[16] *Flask-WTF*. 2013. URL: https://flask-wtf.readthedocs.io/en/1.2.x/.

[17] Maxime Franco and Corrado Lipani. "Automated monitoring of bat species in Belgium". MA thesis. Ecole polytechnique de Louvain, Université catholique de Louvain, 2020. URL: http://hdl.handle.net/2078.1/thesis:25168.

[18] Matthew Frazier. *Flask-Login*. 2013. URL: https://flask-login.readthedocs.io/en/latest/.

[19] *Geospatial Queries*. URL: https://docs.mongodb.com/manual/geospatial-queries/.

[20] Pierre Geurts, Damien Ernst, and Louis Wehenkel. "Extremely Randomized Trees". In: *Machine Learning* 63 (Apr. 2006), pp. 3–42. DOI: 10.1007/s10994-006-6226-1.

[21] Margherita Grandini, Enrico Bagli, and Giorgio Visani. *Metrics for Multi-Class Classification: an Overview*. 2020. arXiv: 2008.05756 [stat.ML].

[22] Stefan Kahl et al. "BirdNET: A deep learning solution for avian diversity monitoring". In: *Ecological Informatics* 61 (2021), p. 101236. ISSN: 1574-9541. DOI: https://doi.org/10.1016/j.ecoinf.2021.101236. URL: https://www.sciencedirect.com/science/article/pii/S1574954121000273.

[23] katspaugh. *Wavesurfer.js*. 2012. URL: https://wavesurfer-js.org.

[24] Nick Littlestone and Manfred Warmuth. "The weighted voting algorithm". In: *Information and Computation 108, 212-261* (1994).

[25] Oisin Mac Aodha et al. "Bat Detective - Deep Learning Tools for Bat Acoustic Signal Detection". In: 2018.

[26] Dirk Merkel. “Docker: Lightweight Linux Containers for Consistent Development and Deployment”. In: *Linux J.* 2014.239 (Mar. 2014). ISSN: 1075-3583. URL: http://dl.acm.org/citation.cfm?id=2600239.2600241.

[27] Muse Group. *Audacity*. Version 3.5.1. URL: https://www.audacityteam.org/.

[28] *Natagora*. URL: https://www.natagora.be/qui-est-natagora.

[29] Vigie Nature. *Vigie Chiro - Monitoring of bats in France*. URL: https://www.vigienature.fr/fr/chauves-souris.

[30] Mark Otto and Jacob Thornton. *Bootstrap*. 2021. URL: https://getbootstrap.com/.

[31] Niels Provos and David Mazières. “A Future-Adaptable Password Scheme”. In: *1999 USENIX Annual Technical Conference (USENIX ATC 99)*. Monterey, CA: USENIX Association, 1999. URL: https://www.usenix.org/conference/1999-usenix-annual-technical-conference/future-adaptable-password-scheme.

[32] *Pymongo: The Python driver for MongoDB*. 2009. URL: https://pymongo.readthedocs.io/.

[33] Armin Ronacher. *Flask*. 2010. URL: https://palletsprojects.com/p/flask/.

[34] Ask Solem. *Celery: Distributed Task Queue*. 2009. URL: https://docs.celeryproject.org/.

[35] *Werkzeug*. 2007. URL: https://werkzeug.palletsprojects.com/en/3.0.x/.

[36] David H Wolpert. “Stacked Generalization”. In: *Neural Networks* 5.2 (1992), pp. 241–259.

[37] *Xeno Canto*. URL: https://www.xeno-canto.org/.

[38] R.D. Zinck. *BattyBirdNET - Bat Sound Analyzer*. 2021. URL: https://github.com/rdz-oss/BattyBirdNET-Analyzer.

[39] R.D. Zinck. *BattyBirdNET-Pi: Automated real-time bat detector*. 2022. URL: https://github.com/rdz-oss/BattyBirdNET-Pi.

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain
Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | **www.uclouvain.be/epl**