SIT378 SCRIPT

1. A bit about history
2. A bit about the system and the components
3. A bit about integration and EchoNet
4. A section with animation (Michael will provide) around message flows between components
5. A brief demo of the HMI running with detections coming up
6. A bit about history

Project Echo is helping nature around the world by creating state-of-the-art bioacoustic tools (audio classifiers) to non-invasively locate, track and monitor endangered species and their predators in their natural environment. It aims to support conservationists. The concept contains a grid of audio sensors

(microphones) are distributed in the natural environment to collect sounds of passing animals. These recorded sound samples are sent back to a central processing server, where an AI-based audio classification model automatically classifies performance and records vital statistics in a local database.

Project Echo has three core values: Voice, Data and Environment. In keeping with the name of the project, Echo, and our commitment to sound, each project member is encouraged to use their own voice and contributors to move the project forward. Project Echo members are passionate about data and strive to find solutions that benefit the environment through data.

1. A bit about the system and the components

The project consists of six components.

**Echo Prototyping**

Echo Prototype is a prototype classification engine that demonstrates that the data processing pipeline can process data from raw, labeled sound clips to species classification. This prototype engine is a series of Jupyters that must be manually executed sequentially (to drive the data pipeline) It will be in the form of a notebook file. These prototype notebook files are expected to provide an environment for data scientists to further develop and refine. A model that improves classification performance in the future.

The Echo prototype is packaged into a Python package for command-line deployment. In this way, the prototype behaves similarly to the way openai's "Whisper" model was deployed, which first inspired the approach to this project.

**Echo Engine**

The Echo Engine component receives requests from the API in the form of sound clips, executes the classification model, and returns the classification results (target labels) to the API for distribution to the Echo HMI endpoint. The classification model is trained offline and deployed to the Echo Engine once the model's performance is validated.

**Echo API**

The Echo API provides a secure endpoint service for remote applications to process sound clips and return classification results. The echo API uses the echo engine to do the work for you. The API also transfers the data to the echo store and retains the sound clips for further investigation.

**Echo Simulator**

Echo Simulator provides a tool to randomly generate animal sound events with random latitude and longitude metadata. These events are passed to the Echo API to process and identify species from sounds. Events are recorded to the Echo Store and propagated to the Echo HMI to visualize detection events on the map in real time.

**1**

**Echo HMI**

The Echo HMI provides a front-end user experience (UX) for customers to interact with all five key components of Echo-net running end-to-end. The HMI renders simulated animals and their vocalizations on a high-resolution satellite map of Otway in real-time via live mode or in a date range of your choice via historical data queries. It also allows the map layer to be filtered by species type and status, providing biodata for 310 species living in the Otways region.

**EchoStore**

EchoStore is a modern database solution for storing raw audio files requested via the Echo API service. The store processes recordings of sound sample requests and provides an opportunity for researchers to later label these records and further fine-tune the quality of the species classification model.

The following diagram shows how these components interact. Note that the Echo Prototype component is only used to develop the model, so it is not explicitly shown here.

1. A bit about integration and EchoNet

The echo Component directory contains all the core production components of the echo system. These components can be instantiated locally on a developer machine using Docker. When an echo component is instantiated in this way, the project's echo team calls it an "EchoNet" environment. Within Docker, a private network called EchoNet is set up, and all component containers participate in this network.

To define the EchoNet environment, each component has a Docker file that describes how to build containers. Most containers read the requirements.txt file to set up the associated Python environment. Because each container is isolated from all others, each component is free to define its own dependencies (for example, its own version of Python).

There are several containers defined by the open source community that form part of EchoNet . This includes the MongoDB and MQTT-Server components. They provide database endpoints and Message Queuing capabilities that facilitate communication between components .

1. A section with animation (Michael will provide) around message flows between components

About exchanging messages between components

Var 1

Simulator sends SensorAudio Msg to Engine. Engine that receives SensorAudio Msg from Simulator sends AnimalSpecies Msg to Store. Simulator also sends AnimalMovement Msg to Store. These inbound and outbound exchanges are JSON Real Time Messages.

HMI sends (API) QueryRequest to Store. The Store that receives the (API)QueryRequest sends an (API)QueryResponsn to the HMI. The above transmission/reception is JSON API.

Finally HMI sends Start/Stop/Reset Msg to Simulator.

Var2

Animal Movement Querry makes a request from the HMI to the API, and the API that receives the request accesses the Database and receives the information and sends it to the HMI. Similarly, Animal Detection Query communicates from HMI to API, API to Database, and API to HMI. Similarly, Animal Vocalisation query communicates from HMI to API, API to Database, and API to HMI. Similarly, Microphone List Querry communicates from HMI to API, API to Database, and API to HMI.

Simulotor Control communicates from the HMI to the API and displays the results in the Simulator via MQTT.

Animal Movement accesses the Database via API from Simulator.

Animal Vocalisation communicates from Simulator to Engine via MQTT and Access Database from Simulator via API.

Animation Detection accesses the Database from the Engine via the API.

Access the Database via API from the Microphone List.

The Animal Species List is a two-way exchange between Engine and GCP, and between Simulator and GCP.

Species Audio Sample has both sides of the interaction between Simulator and GCP.

Var3?

Animal Movement Querry makes a request from the HMI to the API, and the API that receives the request accesses the Database and receives the information and sends it to the HMI. And when you receive the Animal Movement Querry, Animal Movement accesses the database via the API from the Simulator.

Similarly, Animal Detection Query communicates from HMI to API, API to Database, and API to HMI. When the Animal Detection Query is received, Animation Detection accesses the Database via the API from the Engine.

Similarly, Animal Vocalisation query communicates from HMI to API, API to Database, and API to HMI. Then, when the Animal Vocalisation Query is received, Animal Vocalisation communicates from the Simulator to the Engine via MQTT and accesses the Database from the Simulator via API.

Similarly, Microphone List Querry communicates from HMI to API, API to Database, and API to HMI. Then, when the Microphone List Query is received, Access the Database via the API from the Microphone List.

Simulotor Control communicates from the HMI to the API and displays the results in the Simulator via MQTT.

The Animal Species List is a two-way exchange between Engine and GCP, and between Simulator and GCP.

Species Audio Sample has both sides of the interaction between Simulator and GCP.

1. A brief demo of the HMI running with detections coming up