ZIGGS - Anomaly Detection at Home Based on Sound Systems

Jan Imhof
Computer Science Dep.
Karlsruhe Institute of Technology
Karlsruhe, Germany
uwxdb@student.kit.edu

Ivan Milosavljevic
Computer Science Dep.
ISEP School of Electronics
Boulogne-Billancourt, France
ivmi61663@eleve.isep.fr

Nicolas Busquet
Computer Science Dep.
ESGI School of CS
Nanterre, France
nrb.busquet@gmail.com

Adrian Obermuehlner
Computer Science Dep.
ZHAW Zurich University
Zurich, Switzerland
aobermuhlner@gmail.com

I. Introduction

A. Motivation

In today's fast-paced world, ensuring the safety and security of our homes and loved ones has become more critical than ever. Whether it is monitoring elderly family members living independently, ensuring the safety of children, or keeping track of pets when left alone, there is a growing demand for intelligent home monitoring solutions that go beyond basic video surveillance. Traditional security systems often fail to detect nuanced or specific audio-based events such as sudden loud noises, breaking glass, or periods of unusual silence, which can be indicators of emergencies or safety concerns.

The development of an acoustic abnormality detection system utilizing machine learning and IoT devices fills this crucial gap in the market. This innovative software will enable continuous monitoring of environmental sounds through LG's IoT devices, which are already well-integrated into modern smart homes. The use of machine learning allows the system to detect and analyze abnormal audio patterns, such as an elderly person calling for help, a child in distress, or even a pet knocking over an object. Unlike traditional static systems, the machine learning component will learn and improve from new sound data recorded within each household, tailoring its response to the specific acoustic environment of every home.

By integrating this solution with LG's ecosystem of IoT devices, users will have access to a comprehensive, real-time sound monitoring system that adapts to their unique needs. The system will send timely alerts to their mobile devices, allowing them to respond immediately to any potentially dangerous or unusual situations. This adaptive learning capability, combined with the flexibility of using various LG IoT devices such as smart speakers, cameras, or even home appliances, ensures that the system provides an enhanced, personalized safety net for users.

The primary use cases for this system include elderly care, where rapid response to abnormal sounds could prevent accidents or health emergencies; child safety monitoring, where sudden loud noises or periods of unusual silence could indicate a problem; and pet monitoring, where the system can alert owners to disturbances caused by their pets. As homes become more connected, the need for advanced audio-based monitoring systems is increasingly evident, making this software a valuable addition to the market.

B. Problem Statement

Current home monitoring systems fail to effectively detect crucial audio cues like breaking glass or sudden loud noises, often resulting in missed events or frequent false alarms. These systems also lack the ability to adapt to the unique sound environments of individual households, making them unreliable for families with elderly members, children, or pets.

Our solution addresses this by using machine learning and IoT devices to continuously monitor sounds and detect abnormalities. Through user feedback, the system will refine its model over time, adapting to each household and reducing false alarms, ensuring timely and accurate alerts.

C. Related Software

SimpliSafe: This is a well-known home security app that pairs with its proprietary camera and detection systems. While primarily focused on video monitoring, SimpliSafe uses various sensors, such as motion detectors and entry sensors, to identify potential intrusions. Sound detection is used in a specialized way, mainly for detecting the sound of breaking glass, which is a common indicator of a break-in.

ASTRA: This app is described as "state of the art" for anomaly detection, with a focus on video data. ASTRA uses advanced algorithms and artificial intelligence to identify unusual activities through video feeds, which can include identifying abnormal movements or behaviors. Its strength lies in video anomaly detection, making it suitable for environments where visual analysis is more critical than audio.

Edge Impulse: Edge Impulse provides a platform that integrates machine learning with IoT devices, focusing on real-time anomaly detection for environmental and acoustic monitoring. Their system is highly adaptable and has been deployed in various use cases, from smart home appliances to wearables. They emphasize ease of use, allowing developers to collect audio data, train models, and deploy them on IoT devices for real-time detection.

Echo-Guard: This system focuses on detecting anomalous sounds in smart manufacturing environments. It uses acoustic sensors combined with machine learning to monitor the

sounds produced by machinery and detect abnormal events, such as equipment failure or unexpected noises. Its use of convolutional neural networks (CNN) helps to convert audio data into spectrograms, enabling the system to classify and detect anomalies in real-time.

II. Role Assignments

TABLE I
ROLE ASSIGNMENTS

Role	Name	Task Description
AI Developer	Jan Imhof	Specializes in designing and optimizing machine learning algorithms to identify abnormal sounds in a home environment. Responsible for training the model using diverse audio data, allowing it to recognize and adapt to different household sound profiles, such as elderly distress calls, child-related noises, or pet disturbances. Ensures continuous learning from new data, refining accuracy and effectiveness in detecting potential emergencies.
Back-end Developer	Ivan Milosavljevic	Responsible for the development and integration of the back-end architecture that connects LG's IoT devices with the acoustic monitoring system. This includes managing the database to store audio data, implementing security protocols for user authentication, and building APIs for smooth data transmission and real-time alerts. Oversees scalability to handle continuous audio streams and machine learning updates.
Front-end Developer	Adrian Samuel	Focuses on designing and developing the user interface (UI) and user experience (UX) for the mobile and web platforms that enable users to monitor sound-based events in their homes. Ensures seamless interaction between users and the system, including configuring alerts and viewing real-time audio analysis. Key tasks include testing, debugging, and refining the interface for ease of use across various devices.
Front-end Developer	Nicolas Busquet	Oversees the visual design and interaction elements of the user-facing components, ensuring that users receive notifications and alerts in a visually engaging and intuitive manner. Works with tools like Figma to create clean, responsive designs that facilitate quick responses to detected abnormalities, tailored for users with specific needs such as elderly care or pet monitoring.

III. Requirements Analysis

A. User Account Management

1) Sign Up: **Context:** The sign-up screen serves as the main entry point for new users to create their accounts. The login functionality is integrated into the same screen, allowing both new registrations and returning users to access their accounts seamlessly.

Information Required for Sign Up:

- **Email:** The user's email address will be used for sending alerts, notifications, and other important communications. It is also necessary for account recovery and validation.
- **ID/Username:** Users must create a unique identifier for their account, up to 12 characters long. This ID is used to recognize the user within the system and provides an alternative to using their email for login.
- **Password:** Users must create a secure password that meets the following criteria:
  - Minimum of 8 characters
  - Includes lower-case and upper-case letters
  - Contains numbers and symbols to enhance security
  - Passwords are encrypted using SHA256 to protect against data breaches and ensure that login data is not stored in plain text.
- **Personal Information:** Users are required to provide their first and last names, phone number, emergency phone number, and postal address. This information helps ensure that in the case of an alert that isn't acknowledged promptly, emergency contacts can be reached.

Email Validation:

- After users complete the registration form, a verification code is sent to the provided email address.
- The user must enter this code to confirm their email before completing the registration. This ensures the email is valid and accessible by the user.

Redirection:

- Upon successful registration, users are automatically redirected to the login page to access their account using their new credentials.

2) Login: **Context:** The login page is the main gateway for returning users to access their accounts and monitor their home environment. It ensures that only authorized users can access the system.

Required Information for Login:

- **Email or Username/ID:** Users can log in using either their registered email or their unique username.
- **Password:** The user's password is required for authentication. The system verifies the encrypted password stored in the database.

Password Recovery:

- If a user forgets their password, they can request a password reset link via email.
- A code or link is sent to their registered email, allowing them to set a new password following the same security requirements.

3) Administrator Back-office: The website is maintained by admin users. These admin users have access to the back-office part of the website from where they can manage the different items of it (e.g. user accounts, feedback tickets...)

User accounts management

- **Display users:** The administrator have access to a screen which display all the users account created. It displays the information of each user except the confidential ones. The accounts can be sorted by name (alphabetical) or date of creation or of birth. It is also possible to search a user from a search bar.

- **Delete users:** The administrator has the right to delete a user account. It should be used only follow on from a user request.
- **Manage user password:** The administrator is able to send an email for resetting the password of a user.
- The administrator is able to block the access to the feedback ticket system to a user. It is necessary in case of spamming, flooding or disrespect from a user.

Feedback tickets management

- **Display tickets:** The administrator have access to a screen which display all the feedback tickets sent by the users. The tickets can be sorted by date or alphabetical order. They can also be filtered by user, date or status. It's possible to open the ticket page by clicking on it.
- **Answer tickets:** The administrator can answer a ticket from its page. While answering, it is possible to change the ticket status. When a ticket is answered, its author is notified by email.
- **Ticket status:** A ticket can have 4 different status. When received, the status is set to *unread* by default. On the ticket page, the administrator can change the status to *in process*, *solved* or *cancelled*. The *cancelled* status is made for tickets which are not requests or not conform to ethics.

B. User Dashboard

Once logged in, users have access to a personalized dashboard where they can monitor alerts, manage their account information, and interact with the system.

Alerts History:

- Users can view all alerts that have been triggered by the system. These are presented in a tabular format for easy review.
- **Layout:** The table includes columns for the date, time, type of alert, and a brief description of the event. Users can sort or filter alerts based on specific dates, types of events, or alert status (e.g., resolved or pending).
- Users can click on any alert to see more details, such as the audio snippet that triggered the alert, and provide feedback if the alert was accurate or a false positive.

Account Management:

- Users can update their personal details, such as their email, phone numbers, or postal address, directly from their profile settings.
- Users have the option to change their password or update their emergency contacts as needed.
- **Account Deletion:** Users can request to delete their account permanently. This action will remove all personal information, alert history, and preferences from the system. A confirmation step is included to prevent accidental deletions.

C. Feedback Ticket System

- Users can send feedback tickets to the administrator from a specific page. There are 2 kinds of tickets : *bug reports* or *general request*.
- When submitting a ticket, users can provide a brief description of the issue, attach screenshots if necessary, and specify the time when the issue occurred if there's one.
- General requests are tickets which are not related to a technical issue. For example, a request for deleting the account or an idea to improve the website.

D. Alerts and Notifications

Mobile Notifications: Alerts are sent directly to the user's smartphone via push notifications. These alerts include a brief description of the detected anomaly (e.g., "Loud crash detected in the living room") and the time of occurrence. A link to view or listen to a recorded snippet of the audio is also included, enabling users to assess the situation quickly.

Email Alerts: For users who may not have access to their mobile app at all times, email notifications are sent. These emails contain details of the anomaly and a link to access the user portal for further investigation.

SMS Alerts: If configured, SMS alerts can be sent to the primary phone number or emergency contact provided during registration. This ensures that users receive alerts even in areas with limited internet connectivity.

Emergency Escalation: If an alert is not acknowledged or responded to within a predefined time frame, the system can automatically escalate the alert to an emergency contact. This feature is particularly useful in scenarios like elderly care or child monitoring, where immediate attention is critical.

E. Dataset Requirements

1) Normal Data Requirements: The model requires a dataset that captures typical sounds in child play environments to accurately represent normal conditions.

- The dataset should include audio from natural, unstructured settings where children are playing, such as kindergartens, homes, or therapy centers.
- The sounds should cover a range of common activities and interactions, reflecting realistic soundscapes in child play environments.
- Audio quality should be sufficient to capture subtle environmental sounds without excessive background noise.
- The dataset selected for these requirements is the ChildPlayGaze Dataset, which provides realistic audio accompanying children's play and social interactions.

2) Abnormal Data Requirements: To train the model to recognize unusual or potentially hazardous sounds, the dataset must include a variety of abnormal audio events.

- Abnormal sound events should include loud noises such as crashes, glass breaking, and other unexpected sounds that deviate from typical child play environments.
- These sound events should be clearly labeled to facilitate both supervised and unsupervised anomaly detection.
- Data should be varied to represent a wide range of potential abnormal events, helping the model generalize to new or unknown anomalies.

3) Preprocessing Requirements: Before using the dataset for model training, audio data must be preprocessed to standardize inputs and improve model performance.

1) **Audio Trimming**
   - Audio clips should be limited to a uniform length (e.g., 5 seconds) to ensure consistent input dimensions.
   - Trimming is essential to focus on the relevant parts of each sound event and reduce variability in input length.
2) **Mel Spectrogram Conversion**
   - Audio clips must be converted into mel spectrograms, which represent the frequency content of the sound in a way that aligns with human auditory perception.
   - The spectrograms should be stored in a format suitable for machine learning models, such as `.npy` files.
   - Parameters for mel spectrogram generation should be standardized across all audio files to maintain consistency in the input features.

IV. System Specifications

A. Supervised Learning Requirements

- The model must be able to classify sounds based on labeled data.
- It should distinguish between normal and abnormal sounds in various environments.
- The system is expected to identify abnormal sounds, such as breaking glass or loud noises, and categorize them accordingly.

B. Unsupervised Learning Requirements

- The model should learn typical patterns in normal sound environments without relying on explicit labeling.
- It must be able to detect deviations from these normal patterns as potential anomalies.
- The system should recognize unfamiliar or novel sounds that may indicate abnormal conditions.

C. Model Personalization and Feedback

- After deployment, feedback from users (e.g., parents) on anomalies will be collected.
- This feedback will help the model refine its detection capabilities.
- The system will be personalized to each household's unique sound environment.

V. Design and Development Environment

A. Choice of Software Development Platform

TABLE II
Technology and language justification

Technology	Usage	Justification
Linux VPS (on AWS)	Web server hosting	• Easier to configure than a Windows Server • **Light OS**: Make it easier to deploy and leave more space for the website. • **Fast File System**: Reduce the delay of response of the server because the OS read files faster. • **Technologies availability**: Most software are available on Linux.
Docker	Deployment, Containerization	• **Better scalability**: Dockerize services composing our software make easier hardware and software upgrades. • **Better interoperability**: Containers can be deployed on different OS without causing a bunch of issues. • **Service Management**: With every services in a container we can update, stop or restart them easily.
Spring Boot 3 (Java)	back-end Development, API Management, Security	• **Microservice Architecture**: Supports modular development, crucial for building scalable and maintainable systems. • **API Development**: Simplifies REST API creation, enabling seamless communication with the front-end. • **Built-in Security**: Offers authentication and authorization mechanisms like OAuth2 and JWT, enhancing security for sensitive user data. • **Database Integration**: Easily connects to SQL/NoSQL databases, essential for managing data such as alerts, user profiles, and sound analysis logs. • **Java**: Spring Boot uses Java, which is highly stable, widely supported, and suitable for enterprise-level applications, providing robust back-end performance.
Vue.js 3 (TypeScript)	front-end Development, Real-time User Interface	• **Lightweight & Fast**: Minimal footprint ensures fast loading times, ideal for responsive user interfaces on both mobile and web platforms. • **Reactive Components**: Real-time data binding allows users to receive alerts instantly without refreshing the page. • **Component-Based Structure**: Enables the creation of reusable UI elements for dashboards, alerts, and login forms. • **TypeScript**: Using TypeScript with Vue.js provides type safety, reducing runtime errors and making the front-end codebase more maintainable and scalable.

B. AI Development Platform

This project aims to develop an AI model for detecting anomalies in audio data. The development is performed using **Google Colab**, which provides a cloud-based environment with access to an NVIDIA Tesla T4 GPU at no cost. The model's code, data, and outputs are stored on Google Drive for easy access and sharing.

1) Platform and Resources:

- **Platform**: The development environment is Google Colab, a cloud-based platform offering free GPU resources.
- **Hardware**: The provided NVIDIA Tesla T4 GPU meets the project's computational requirements.
- **Storage**: Audio files and generated mel spectrograms are stored on Google Drive, allowing for easy file sharing and collaborative work.

2) Programming Language: **Python** is used as the primary programming language due to its versatility and widespread adoption in machine learning. Python offers extensive libraries, making it highly suitable for one-time preprocessing tasks and deep learning model development. For this project:

- The initial codebase was based on an existing audio anomaly detection notebook, which can be accessed at the following link: Audio Anomaly Detection Notebook.
- The framework was modified from `Keras` to `PyTorch` for compatibility with modern practices, as `PyTorch` has become more prevalent in the AI research community and the AI developer is more familiar with `PyTorch`.

3) Software Versions: To ensure compatibility and reproducibility, the following versions are used in this project:

- **Python**: 3.10 (Colab default)
- **CUDA**: 12.2 (for GPU acceleration)
- **Torch**: 2.5.0+cu121
- **Torchaudio**: 2.5.0+cu121

C. Model Deployment

The deployment of the AI model for anomaly sound detection is planned to be executed on Amazon Web Services (AWS) using a serverless approach. This strategy uses AWS Lambda and API Gateway to host the model, ensuring scalability, cost efficiency, and minimal infrastructure management. By utilizing Lambda, the model is loaded and served on-demand, allowing for quick responses to inference requests without the need for a continuously running server. This approach is chosen over AWS SageMaker because SageMaker introduces significant overhead and is primarily designed for high-throughput or continuous workloads. Given the relatively low frequency of model calls, SageMaker would add unnecessary complexity and pose a risk of escalating costs due to the complexity.

1) Platform and Resources:

- **Platform**: The deployment environment will be AWS Lambda, using a custom Docker container to package the model and all necessary dependencies.
- **Containerization with Docker**: The model and dependencies (PyTorch, NumPy, etc.) are packaged into a Docker container image, ensuring consistent runtime behavior. This container allows AWS Lambda to execute the model with all required libraries, avoiding compatibility issues.
- **Hardware**: AWS Lambda functions dynamically allocate resources based on workload requirements. For this project, a CPU-based instance is sufficient given the expected low call frequency, reducing both cost and complexity.
- **Model Format**: The model is saved in .pt format (PyTorch model format), compatible with the Lambda function configured to use PyTorch libraries within the Docker container.

2) Programming Language and Environment: The deployed model will continue to use Python as the programming language, given its compatibility with PyTorch and widespread support in AWS Lambda.

3) Software Versions and Dependencies in Docker: The Docker container provides a consistent environment for the Lambda function, ensuring compatibility and reproducibility:

- Base Image: public.ecr.aws/lambda/python:3.8 (AWS Lambda Python 3.8 base image).
- Dependencies: The Dockerfile includes the installation of PyTorch, NumPy, and other required libraries.
- Python Version: 3.8, as provided by the AWS Lambda base image.
- Torch and Torchaudio Versions: Compatible versions are specified in the Dockerfile to ensure consistency with the .pt model format.

VI. Cost Estimation

A. AI Model Development

This project utilizes free resources provided by Google Colab and Google Drive:

- **Software Cost**: Free, as Google Colab and Google Drive offer no-cost access for the project's requirements.
- **Hardware Cost**: None, as the Tesla T4 GPU is provided within Google Colab's free tier.

B. AI Model Deployment

For this serverless deployment, AWS charges are based on the compute time used per request in Lambda and the request volume handled by API Gateway:

- AWS Lambda with Docker: The cost is based on compute time (measured in milliseconds) per request. Given the lightweight nature of the model and expected low frequency of a few calls per day, Lambda expenses are anticipated to be minimal, approximately 0.50–1 Dollar per month.
- API Gateway: This service charges per million requests. With a low request frequency, monthly costs for API Gateway are expected to be negligible, estimated at around 0.10–0.50 Dollar per month.
- Total Cost: The combined costs for AWS Lambda and API Gateway for a few daily calls are estimated at approximately 0.60–1.50 Dollar per month.

VII. Implementation and Integration

A. Login System

The login system allows users to securely access their accounts using their email and password. Users can also opt for a password recovery mechanism through email verification. Spring Boot handles back-end processes, including password encryption using SHA-256 and validation against the database. Vue.js ensures the front-end provides an intuitive and seamless experience for users, with error handling and clear feedback messages on incorrect login attempts or successful authentication.

B. Administrator Back-office

Administrators can manage user accounts and feedback tickets through a dedicated back-office interface. The back-office allows viewing, sorting, and filtering user accounts and ticket status. This module leverages Spring Boot for back-end operations and Vue.js for front-end management tools.

C. User Dashboard

The dashboard gives users access to view historical alerts, manage personal information, and configure system settings. It features filters to sort alerts by type and date, allowing users to analyze past events. The front-end, developed with Vue.js, ensures responsive design across devices. Spring Boot manages the back-end, ensuring secure communication and data retrieval from the database.

D. Feedback tickets sending

A specific interface allows users to send feedback to the administrators through tickets. This interface would have some basic text editing features such as italic font, bold font, etc. This interface is made with Vue.js. On server side, this feature needs a few API end-points (to create and delete) and a table in the database.

E. Alerts and Notifications

The system provides multi-channel notifications through push notifications and emails when abnormal sounds are detected. Vue.js powers the user interface where users can configure alert preferences. Spring Boot manages the back-end operations for sending alerts and handling the escalation process in case of unacknowledged alerts.

F. Real-time Sound Monitoring

The back-end, built using Spring Boot, communicates with the machine learning models, which detect abnormal sounds such as breaking glass, loud noises, or sudden silence. Python models, integrated through APIs, perform sound analysis and anomaly detection, ensuring efficient real-time monitoring.

G. Machine Learning Integration

Machine learning models play a vital role in anomaly detection. Supervised models identify patterns based on pre-labeled audio data, while unsupervised learning models recognize new and unusual sounds by learning typical household environments. These models are trained using Python and integrated into the system via APIs provided by the Spring Boot back-end.

H. Database Management

The system uses PostgreSQL to store user information, alert history, and sound data. Spring Boot handles database communication, ensuring data integrity and security. User profiles, system alerts, and other data are stored relationally, facilitating efficient querying and data retrieval. Regular backups and encryption protocols ensure data safety and compliance with privacy policies.

I. Data Implementation Process

This section details the step-by-step implementation of the audio preprocessing.

1) Audio Data Collection and Duration Normalization: The audio data preprocessing begins with loading WAV files from a designated directory. The audio files are processed to ensure each file meets the required duration of exactly 5 seconds. If a file is shorter than 5 seconds, additional audio files are concatenated to reach the duration limit.

- **Identify WAV files**: List all WAV files in the target directory, excluding specified subfolders.
- **Calculate Duration**: For each audio file, calculate the duration in seconds to determine if it meets the 5-second requirement.
- **Combine and Trim Audio Files**:
  - Concatenate audio files until the total duration is at least 5 seconds.
  - Trim the combined audio to exactly 5 seconds if it exceeds the limit.
  - Save the trimmed file in a separate output directory, preserving folder structure.

2) Mel Spectrogram Generation: Each trimmed audio file is converted into a mel spectrogram, which provides a time-frequency representation of the sound.

- **Load Audio Files**: Load each WAV file from the output directory created in the previous step.
- **Convert to Mel Spectrogram**:
  - Use a set of predefined parameters, such as FFT size, hop length, and number of mel bands, to create a standardized mel spectrogram.
  - Convert the spectrogram to decibel scale to enhance feature visibility.
- **Padding and Trimming**: If the mel spectrogram has fewer frames than expected for a 5-second clip, pad it with zeros. If it exceeds the expected frame count, trim the excess frames to maintain uniform input dimensions.

- **Save as .npy Files**: Save each spectrogram as an `.npy` file in a structured directory to facilitate easy access during training.

VIII. AI Implementation Process

The following section describes the main stages involved in implementing the AI model for audio anomaly detection under the development environment on Google Colab with Python and PyTorch.

A. Data Preparation and Loading

- **Data Collection**: The mel spectrograms for the audio files are stored in Google Drive and are structured into folders based on the type of sound (e.g., normal or anomaly).
- **Dataframe Creation**: A function is used to load the mel spectrograms and create a structured dataframe, labeling each entry based on sound type and directory (train or test). The data is split so that 15% of samples are used for testing, and the rest are used for training.

B. Data Generator

The data generator function is responsible for loading data in batches during training and validation:

- **Shuffling and Batch Creation**: To avoid overfitting and ensure variety, data is shuffled and loaded in small batches. Each batch contains both the mel spectrograms and their corresponding labels.
- **One-Hot Encoding of Labels**: Labels are converted to a one-hot encoded format to support multi-class classification.

C. Model Architecture (CNN with Regularization)

The model is implemented as a Convolutional Neural Network (CNN) with regularization techniques such as dropout and batch normalization to improve generalization. The main components are:

- **Convolutional Layers**: Extract spatial features from the mel spectrograms. Each convolutional layer is followed by batch normalization and pooling.
- **Fully Connected Layer with Dropout**: After flattening, a fully connected layer with dropout is used for additional regularization.
- **Output Layer**: A softmax activation function is applied in the final layer to classify sounds into six categories.

D. Training and Validation Loop

The training function, `fit_and_predict_model`, manages the training and validation of the model:

- **Data Splitting**: The function divides the data into training and validation sets, using 80% for training and 20% for validation within the training dataset.
- **Loss Function and Optimization**: The model is trained using Cross-Entropy Loss for multi-class classification, and the optimizer is Adam with L2 regularization.
- **Training Epochs and Early Stopping**: The model is trained over multiple epochs, and early stopping is used to prevent overfitting by monitoring validation loss.

Fig. 1. Training and Validation Accuracy and Loss Over Epochs

E. Model Evaluation

TABLE III
Classification Report for Machine Learning Model on Various Audio Categories

	Precision	Recall	F1-Score	Support
ChildrenPlay	1.00	1.00	1.00	29
Accidents	0.00	0.00	0.00	1
Car Crash	0.80	1.00	0.89	4
Conversation	1.00	1.00	1.00	2
Gun Shot	0.90	1.00	0.95	9
Scream	1.00	0.83	0.91	6
Accuracy		0.96		51
Macro Avg	0.78	0.81	0.79	51
Weighted Avg	0.95	0.96	0.95	51

Explanation of Terms:

- **Macro Average**: The macro average calculates precision, recall, and F1-score for each class independently and then takes their unweighted mean. This gives equal importance to each class, which can highlight performance on underrepresented classes.
- **Weighted Average**: The weighted average calculates metrics for each class and then takes their average, weighted by the number of samples (support) in each class. This provides a more accurate overall metric when there is class imbalance.
- **Support**: Support represents the number of true instances for each class in the dataset. It shows the distribution of data across different categories and helps in understanding how prevalent each class is in the evaluation.

IX. Architecture Design and Implementation

A. Overall architecture

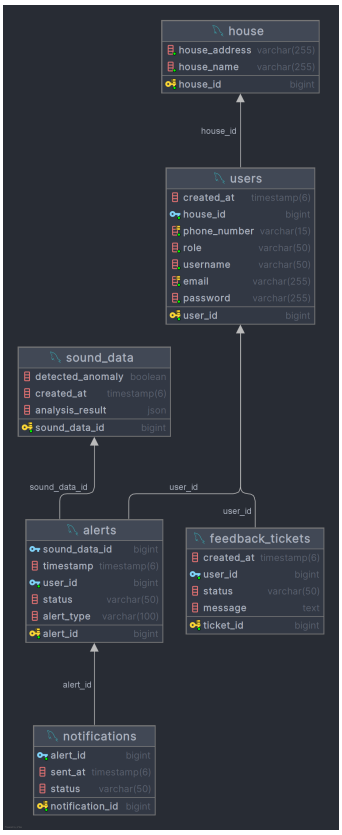

Fig. 2. Database structure

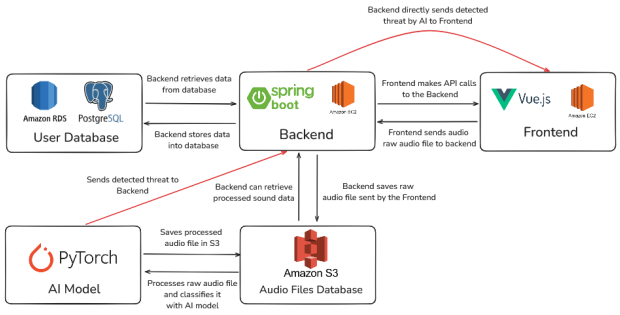

Fig. 3. System Architecture

X. Modules Overview

A. Directory organization Backend

Directory	File Names	Module
Ziggs_Backend/ src/main/java/ com/ziggs/ ziggs_backend/ controller	AlertController.java AuthController.java FeedbackTicketController.java HouseController.java NotificationController.java SecureController.java SoundDataController.java UserController.java	Controller
Ziggs_Backend/ src/main/java/ com/ziggs/ ziggs_backend/ service	AlertService.java FeedbackTicketService.java HouseService.java NotificationService.java SoundDataService.java UserService.java	Service
Ziggs_Backend/ src/main/java/ com/ziggs/ ziggs_backend/ repository	AlertRepository.java FeedbackTicketRepository.java HouseRepository.java NotificationRepository.java SoundDataRepository.java UserRepository.java	Repository
Ziggs_Backend/ src/main/java/ com/ziggs/ ziggs_backend/ entity	Alert.java FeedbackTicket.java House.java Notification.java SoundData.java User.java	Entity
Ziggs_Backend/ src/main/java/ com/ziggs/ ziggs_backend/ dto	LoginRequestDTO.java UserDTO.java UserId.java	DTO
Ziggs_Backend/ src/main/java/ com/ziggs/ ziggs_backend/ config	FirebaseConfig.java WebConfig.java	Configuration
Ziggs_Backend/ src/main/resources	application.properties application.properties.example ziggs-7421a-firebase-adminsdk-2ivcm-8b8925d87b.json	Resources
Ziggs_Backend	Dockerfile	Docker
Ziggs_Backend	pom.xml	Build

TABLE IV
Directory Structure and Modules for Backend

The application is divided into several modules, each with a specific role and responsibility to ensure clarity, modularity, and scalability.

Controller:

This module manages incoming HTTP requests and serves as the entry point for users to interact with the application. It defines endpoints for features such as authentication, notifications, user management, and feedback handling. Requests received by the controllers are passed to the appropriate service layer for processing. By decoupling request handling from business logic, it ensures clarity, modularity, and easier debugging.

Service:

This module contains the core business logic of the application. It processes data received from the controllers, applies business rules, and handles complex operations before interacting with the repositories for data storage or retrieval. For example, the AlertService validates incoming alerts before storing them, and the UserService manages user-related operations like registration and profile updates. This separation ensures maintainability, scalability, and flexibility for adapting to future requirements.

Repository:

The repository module handles data persistence and retrieval. It interfaces directly with the database using JPA/Hibernate, abstracting the complexity of database queries. Each repository corresponds to a specific entity, such as `UserRepository` for user data or `AlertRepository` for alerts, ensuring efficient data access and consistency. This design provides standardized and reusable methods for database operations.

Entity:

The entity module defines the application's data models. It maps Java objects to database tables using JPA annotations, providing a clear and structured way to represent data. Entities such as `User`, `Alert`, and `FeedbackTicket` define the schema and relationships between different data points. This abstraction ensures database interactions remain intuitive and aligned with the business logic.

DTO (Data Transfer Object):

This module structures data for specific use cases, such as API requests and responses. DTOs like `UserDTO` and `LoginRequestDTO` are used to simplify communication between layers by encapsulating only the required fields. They improve performance by reducing unnecessary data transfer and ensure consistent data formats across endpoints.

Configuration:

The configuration module manages external library integrations and application settings. It includes files like `FirebaseConfig` for setting up Firebase authentication and `WebConfig` for defining interceptors or CORS policies. This module ensures seamless scalability and easy customization of application behavior without altering the core logic.

Resources:

This directory contains essential configuration files required for the application to run properly. It includes `application.properties` for defining environment-specific settings (e.g., database connections) and Firebase credentials for integrating authentication services. These files centralize setup details and streamline environment configuration.

Docker:

The Docker module provides the steps necessary to containerize the application. The `Dockerfile` defines how the application is built and run in a Docker container, ensuring consistency across development, testing, and production environments. It simplifies deployment, allows better resource isolation, and enables scalability through container orchestration tools.

Build:

The `pom.xml` file defines the project's dependencies, plugins, and build structure. It ensures compatibility by managing library versions and provides an efficient way to build, test, and deploy the application. This file acts as the foundation for the Maven-based build process.

B. Directory organization Data and AI

Directory	File Names	Module
AI_Model/	Sound_Preprocessing.ipynb Pytorch_Supervised_Model.ipynb	Data Processing / Model Creation
AWS_Backend_Model/	app.py app_api.py model.py CNN_RegDrop.pt	S3 Monitoring and Model Execution / Legacy API
AWS_Backend_Model/	dockerfile	Deployment Configuration

TABLE V
DIRECTORY STRUCTURE AND MODULES FOR AI WORKFLOW

1) Data Processing / Model: This module preprocesses audio data into mel spectrograms, trains a CNN model to classify audio anomalies, and evaluates the model's performance. Detailed explanations can be found in section VIII.

Key Components:

- **Data Preparation**: Converts audio into mel spectrograms and structures datasets.
- **CNN Architecture**: Implements convolutional layers, dropout, and batch normalization for classification.
- **Training and Validation**: Trains the model with cross-entropy loss and evaluates metrics such as precision and accuracy.

2) S3 Monitoring and Model Execution: This module automates anomaly detection by monitoring an S3 bucket for audio files, processing them with a CNN model, and sending notifications if anomalies are detected.

Initial Design: Originally, an API Gateway with AWS Lambda was implemented to provide predictions via a `/predict` endpoint. This design worked well for individual requests, as shown in Figure 7. However, maintaining multiple API endpoints across different backends is too complex for a system where data processing could be centralized.

Current Design: The job-based system in `app.py` (Work in progress):

- Monitors the `input-data/` folder in S3.
- Processes files with a preloaded CNN model (`CNN_RegDrop.pt`).
- Sends notifications for anomalies and moves processed files to `processed-data/`.

3) Dockerfile: This Dockerfile was created for the initial design where the AI model was deployed via an API gateway and AWS Lambda. It uses the AWS Lambda Python runtime base image and is optimized for small size by installing only the necessary dependencies such as PyTorch, Flask and `apig-wsgi`. The application code (`app.py`, `model.py`) and the trained model (`CNN_RegDrop.pt`) are included in the container. The resulting image is pushed to Amazon Elastic Container Registry (ECR) for deployment in AWS Lambda.

C. Directory Organization Front-End

Directory	**File Names**	**Module**
Vue_Frontend/src/	App.vue main.js	Initialization
Vue_Frontend/src/ router	index.js routes.js	Routing
Vue_Frontend/src/ assets/	css/*.css fonts/*.woff fonts/*.woff2 fonts/*.ttf fonts/*.eot fonts/*.svg img/*.png img/*.jpg img/*.svg img/*.gif sass/*.scss	Assets
Vue_Frontend/src/ components/	Button.vue Dropdown.vue FormCompenent.vue LabelledInput.vue ziggTable.vue Inputs/formGroupInput.vue index.js	Vue Components
Vue_Frontend/src/ components/Cards	Card.vue ChartCard.vue StatsCard.vue TableCard.vue	Card Vue Components
Vue_Frontend/src/ components/ SidebarPlugin	SidebarLink.vue SideBar.vue MovingArrow.vue index.js	Sidebar Vue Components
Vue_Frontend/src/ layout	AlarmTab.vue FooterComponent.vue GlobalyLayout.vue HeaderComponent.vue	Layouts
Vue_Frontend/src/ layout/dashboard	Content.vue ContentFooter.vue DashboardLayout.vue MobileMenu.vue TopNavbar.js	Dashboard Layout
Vue_Frontend/src/ pages	Dashboard.vue Maps.vue NotFoundPage.vue Notificiations.vue TableList.vue Typography.vue UserProfile.vue Notifications/ Notification-Template.vue	Pages
Vue_Frontend/src/ pages/UserProfile	components/edit-household-modal.vue components/edit-modal.vue components/invite-modal.vue EditProfileForm.vue HouseCard.vue UserCard.vue	User Profile Components
Vue_Frontend/src/ plugins	globalComponents.js ziggDashboard.js globalDirectives.js	Plugins

TABLE VI
DIRECTORY STRUCTURE AND MODULES FOR FRONT-END

Initialization:

This module is in charge of the initialization of the whole project. It loads the dashboard with Vue and activate the routing part.

Routing:

The routing files create an instance of the Router class and save all the routes (URIs) of our project. It links the routes with the wanted components. The routing is essential to allow the website to have multiple pages.

Assets:

The Assets section is used for no code files such as style sheets and images. The files of the Assets folder are not fixed and may be deleted or created during its lifetime.

Vue Components:

Vue.JS works with components, which are pieces of HTML/CSS and a script language you can implement with other components to create pages. This module Vue Components contains the components used in our pages. Some components use other components to work, these components are grouped in sub-folders.

Card Vue Components:

This section is about the sub-folder "Cards" of the Components module. It contains different kind of Cards components and it is where new Cards components should be stored.

Side Bar Vue Components:

This section is about the sub-folder "SidebarPlugin" of the Components module. It contains the Sidebar component and its sub-components as well.

Layouts:

This module gathers the Layouts components. A layout is a component which defines the position of elements. A layout can be used for different pages. This module can contain sub-folders for layouts which need specific components. These sub-folders contain the layout and its components.

Dashboard Layout:

This sub-folder of the Layouts module is meant to store the Dashboard component and its sub-components.

Pages:

This Pages module contain the pages components of the project. These components may use or complete a layout with more specific information. Pages components are basically the components that are displayed to the user. Some pages may use its own components, in this case the page and its components can be put in the same folder.

User Profile Components:

This sub-folder of the Pages module stores the User Profile page and its sub-components in a folder.

Plugins:

The Plugins module is used to import external libraries and globalize their components to the whole project.

Modules:

Login/Create Account Page

This module handles user authentication and account creation using [specify technology or framework, e.g., Firebase, OAuth]. It supports multiple authentication methods, including email/password and social media logins. Upon successful login or account creation, the user is redirected to the dashboard. User session management is implemented to maintain login states across visits. The authentication process is asynchronous, with progress animations to enhance the user experience during the wait.

Household Management Page

On the Household Management page, users can add and manage household members. This feature allows users to define roles and permissions for each member concerning device access and data visibility. The interface provides forms to input member information and uses [technology or method, e.g., AJAX calls] to submit this data to the server. Changes are updated in real-time using [state management technique, e.g., Redux or Context API], ensuring all household data across the platform remains synchronized.

Devices Page

The Devices page enables users to register and manage devices equipped with microphones. It includes a form to enter device details and a mechanism to connect these devices to the user's network. Each device sends audio data to the backend for processing. The page provides real-time feedback on the device's status, such as connectivity and recording state, and uses asynchronous requests to handle registration and updates without page reloads.

Dashboard Page

The Dashboard serves as a central hub for users to get an overview of their household and device settings. It displays aggregate data from all connected devices, including audio monitoring status and event logs. The dashboard updates dynamically, reflecting changes in real-time through web sockets or polling mechanisms. Widgets and graphical representations (charts, graphs) provide a user-friendly interface to monitor household activities and device statuses.

Popup Alerts

Popup alerts are implemented across all pages to notify the user of any detected abnormalities by the AI in the backend. These alerts are triggered in real-time and are designed to catch the user's attention without disrupting ongoing tasks.

The alerts provide concise information about the nature of the abnormality and prompt the user for immediate actions if necessary. Integration with the backend AI uses [specific technologies, protocols], ensuring timely and relevant notifications.

Backend Integration and AI Processing

This module describes the interaction between the frontend and the AI algorithms running on the server. The backend receives audio data from devices, processes it using machine learning models to detect abnormalities, and sends alerts back to the frontend. API endpoints are designed for efficient data transmission and security. Details include the use of RESTful services or real-time communication protocols like WebSockets for continuous data flow.

XI. Use Cases

A. Backend

The following examples illustrate some use cases for endpoints within the web application. These are just a few among many others, meant to demonstrate how various functionalities are integrated into the system to handle different types of user requests.

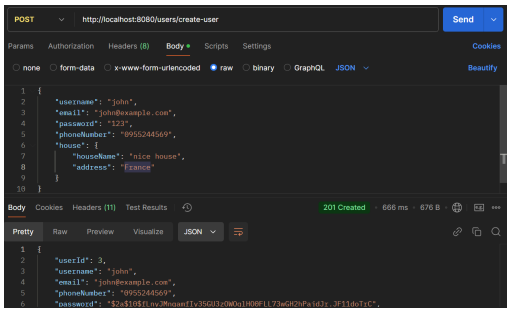

Fig. 4. Endpoint for creating a user. The user enters his information such as username, password, address... The user information is then stored into our relational database.

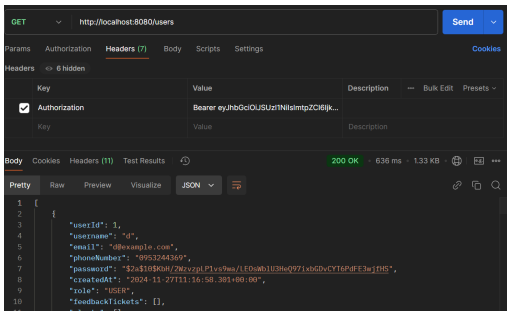

Fig. 5. Endpoint for retrieving user profile data. This endpoint allows the web app to fetch detailed information about a user, such as their name, email...

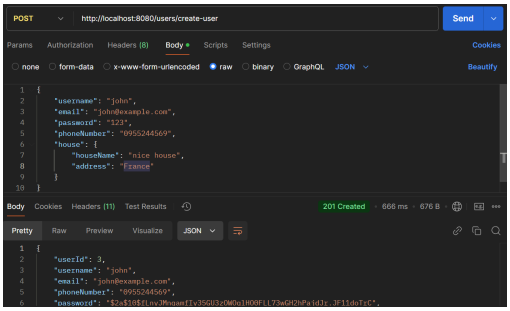

Fig. 6. Endpoint for user login. Upon successful login, the system generates a custom token that grants access to restricted endpoints based on the user's role, while also maintaining the session.

These use cases represent just a fraction of the possible interactions within the application, each designed to handle specific tasks and improve user experience through seamless API communication.

B. AI Model Backend

Figure 7 illustrates the use of an API endpoint for anomaly detection in audio data. A POST request is sent to the endpoint hosted on AWS API Gateway, which provides audio input data in JSON format.

The API processes the input using the provided AI model and returns the following response

- **Anomaly Score**: A numeric value indicating the probability that the input is anomalous.
- **Predicted Class**: The classification result, where 0 means that no anomaly was detected.
- **Predicted Probability**: The confidence value for the predicted class.

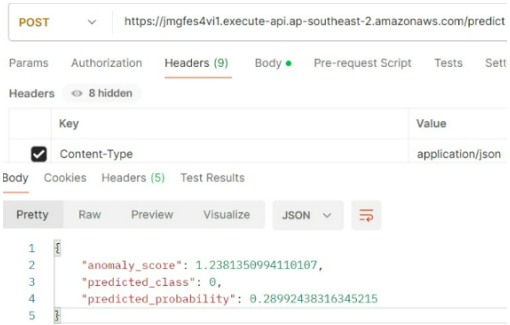

Fig. 7. Postman Test of API Gateway Design

Figure 8 shows the `input-data/` folder in an Amazon S3 bucket. This folder is continuously monitored by the new job-based approach implemented in `app.py`.

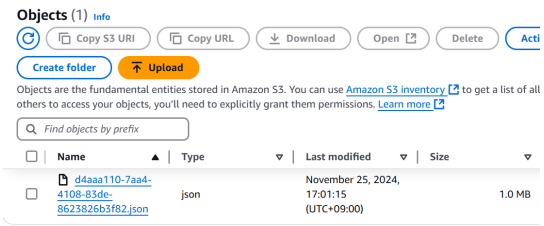

Fig. 8. S3 Bucket that is being monitored

C. Front-End

Figure 9 shows the user dashboard. To access this page, the user just need to login. It is composed of a side menu, the main part and a header. By default, the main part displays the last alerts and notifications. The user can navigate through the dashboard with the side menu.

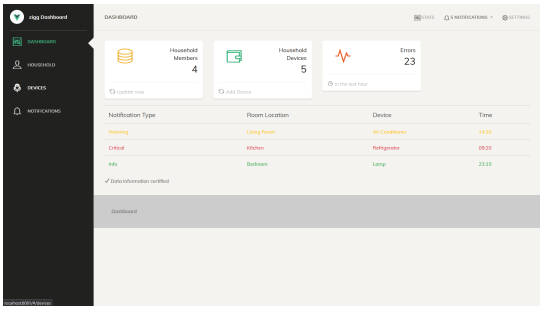

Fig. 9. The user dashboard main page

Figure 10 shows the household section during editing. To access the household page of the dashboard, the user has to click on the ”Household” button in the side menu. At this point, the user can see the household members and information. The ”Edit Profile” button will open the modal you can see on the screenshot. When the form is completed and submitted, the household information are updated.

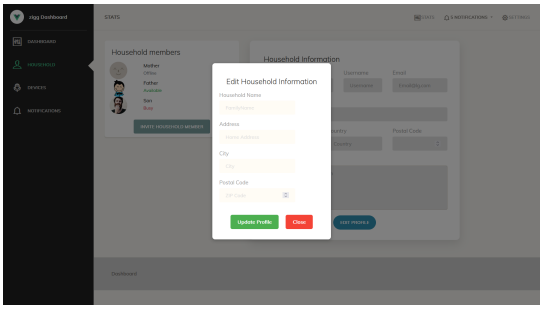

Fig. 10. Editing household’s information

Figure 11 shows the ”Devices” section of the dashboard. This section can be accessed by clicking on the ”Devices” button on the side menu. From this page users are able to add new devices linked to their household or remove the existing devices.

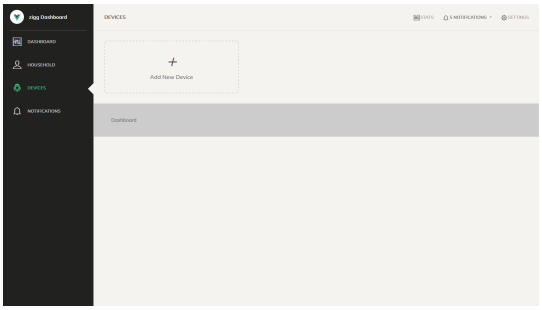

Fig. 11. Devices dashboard