Scientific Research Publishing

Open Journal of Applied Sciences, 2025, 15(8), 2442-2463
https://www.scirp.org/journal/ojapps
ISSN Online: 2165-3925
ISSN Print: 2165-3917

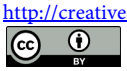
A Machine Learning-Based App for Sound Detection and Speech Recognition for the Deaf and Hard of Hearing

Nada Barnawi, Mohammed Alnuem

Information Systems Department, King Saud University, Riyadh, KSA
Email: Nbarnawi@ksu.edu.sa, Malnuem@ksu.edu.sa

How to cite this paper: Barnawi, N. and Alnuem, M. (2025) A Machine Learning-Based App for Sound Detection and Speech Recognition for the Deaf and Hard of Hearing. *Open Journal of Applied Sciences*, **15**, 2442-2463.
https://doi.org/10.4236/ojapps.2025.158163

Received: July 16, 2025
Accepted: August 22, 2025
Published: August 25, 2025



Abstract

This project uses AI to improve safety and communication for the deaf and hard-of-hearing community in Saudi Arabia. By combining real-time sound detection and speech recognition, it offers alerts for potential dangers and supports communication. The system is built with Android Studio, TensorFlow Lite, and Speech-to-Text APIs. The core model uses YAMNet for feature extraction and a Deep Neural Network (DNN) for classification, achieving 95.77% accuracy. Future updates will include features like customizable alerts, hazard detection, sign language support, and location tracking, aiming to enhance safety and inclusivity.

Keywords

Machine Learning-Based, Deaf and Hard of Hearing, Sound Detection, Speech Recognition, Realtime, Android Application, Assistive Technologies, Saudi Arabia

1. Introduction

Everyday risks, such as responding to fire alarms, can be challenging for individuals who are deaf or hard of hearing, as they often rely on others for alerts, leading to delays in critical situations. With over 430 million people globally affected by disabling hearing loss—expected to surpass 700 million by 2050—this group represents one of the largest disability demographics [1] [2]. Advances in AI and computer science have introduced innovative tools to assist those with hearing impairments, enhancing their awareness of surrounding dangers by recognizing sounds, identifying hazards, and enabling timely responses.

The objective of this research is to develop a mobile application for Android

smartphones that operates in Arabic, leveraging sound detection technology to alert deaf and hearing-impaired individuals to potential warning signals in real time. This app will also facilitate communication with relevant authorities during emergencies by converting spoken conversations into text, thereby enhancing communication effectiveness in the Kingdom of Saudi Arabia. This app called Fatn.

To achieve this, we will utilize sound detection technology based on five types of emergency alarm audio, combining deep learning techniques with YAMNet, a pre-trained neural network known for its efficiency in audio feature extraction. YAMNet employs the MobileNetV1 architecture to classify audio events into 521 categories, processing audio waveforms to generate embeddings, predicted scores, and log mel spectrograms. Its embeddings capture essential features, enabling the development of audio classifiers with minimal labeled data [3]. This study will use YAMNet's calibrated embeddings to enhance the detection and classification of emergency alarms through deep learning and transfer learning techniques. Additionally, the app will integrate speech recognition technology for real-time transcription during calls to authorities. We will utilize the Android speech-to-text API, which captures audio input, analyzes it for speech content, and produces corresponding text output.

This research will leverage a carefully selected collection of 3719 audio samples associated with the five types of emergency alarms. Through rigorous training and evaluation methods, we aim to achieve high accuracy in recognizing and categorizing these alarms based on their acoustic characteristics.

By integrating AI and speech recognition technologies, this app aims to enhance safety, independence, and social integration for individuals with hearing impairments in Saudi Arabia. It also contributes valuable resources for researchers and supports global efforts to improve quality of life for the deaf and hard-of-hearing community.

The key contributions of this research include:

- AI-Powered Sound Detection: Utilizing machine learning algorithms, such as DNNs, for real-time classification of critical environmental sounds.
- YAMNet Utilization: Employing YAMNet as a feature extractor to capture rich characteristics from emergency audio.
- Speech-to-Text Integration: Implementing real-time speech-to-text functionality optimized for Arabic to facilitate seamless communication during emergencies.

This paper is organized into several key sections. Section 2 reviews relevant literature on research and applications. Section 3 focuses on the core technologies, specifically real-time sound detection and speech recognition architectures. Section 4 outlines the system implementation process, detailing the setup in Android Studio, deployment of the sound detection model for emergency alerts, and integration of speech recognition for real-time transcription. Section 5 evaluates the app and discusses the development journey. Finally, Section 6 summarizes the re-

search project's conclusions and outlines future tasks necessary for app completion.

2. Literature Review

This section reviews various applications and research initiatives focused on enhancing the safety and quality of life for individuals who are deaf or hard of hearing. Leveraging technologies such as sound detection and speech recognition, these solutions aim to bridge communication gaps and improve access to critical information.

The Enssat application utilizes Google Glass to aid deaf and hard-of-hearing users by providing sound detection and speech recognition in both Arabic and English. It records ambient sounds through the device's microphone and matches them against a stored sound library using the MusicG library. Enssat also offers real-time speech-to-text transcription via Google's Speech-to-Text service, along with translation features that convert spoken words and images into various languages using Google's Translation API [4]. Another study describes a specialized virtual assistant that categorizes sounds for individuals with hearing impairments, employing deep neural networks trained on the UrbanSound8K dataset. This system includes gesture recognition for translating Indian Sign Language into text and audio, facilitating communication with non-deaf individuals [5]. Saifan's Deaf Assistant Digital System alerts users to significant sounds using vibrations and visual notifications, employing deep learning techniques for accurate sound identification [6]. Additional research discusses sound detection algorithms based on Gaussian Mixture Models (GMMs), highlighting their application in speech recognition and sound classification [7]. The iHelp application aims to provide emergency assistance to deaf-mute individuals, enabling them to report emergencies via SMS without needing internet access [8]. Furthermore, recent advancements in audio classification demonstrate the potential of deep learning for various applications. One study applies YAMNet, a robust deep learning architecture, to identify firearms based on gunshot noises. By employing Mel spectrograms for multi-class audio classification, the researchers achieved a remarkable accuracy of 94.96% using a dataset of 1174 audio samples from 12 distinct weapons. This research emphasizes the effectiveness of YAMNet in forensic and military contexts, highlighting its capability for precise gun type identification [9].

Several mobile applications enhance accessibility for the deaf and hard-of-hearing community. The Sound Alert App detects significant environmental sounds such as doorbells and alarms, integrating seamlessly with existing infrastructure [10]. Similarly, the Deaf and Hearing Impaired (APK) alerts users to loud sounds through vibrations and visual signals [11]. Live Transcribe & Sound Notifications enables real-time transcription in over 80 languages and provides sound alerts for potentially hazardous situations [12]. Rogervoice transforms phone communication for users by offering real-time call subtitles in multiple languages, allowing for independent interaction with others [13]. The "Kulluna Amn" app enhances

public security by allowing users to report incidents, while TapSOS serves as an essential tool for non-verbal emergency communication [14] [15]. Together, these applications and research initiatives illustrate the potential of technology to significantly improve the lives of individuals with hearing impairments.

3. Methodology

Given the research objective of creating an application that alerts users to alarm sounds, this study will delineate three primary stages for implementing the proposed system, as depicted in "Figure 1". The initial stage serves as the foundational block for the app, focusing on integrating the necessary dependencies and requirements for configuration with Android devices using Android Studio. Once this stage is completed, the app is prepared for integration and enhancement with subsequent stages and requirements. The second stage concentrates on constructing the sound detection classification system within the application environment. This encompasses activities such as data collection, preprocessing, feature extraction, model training, real-time sound classification, and issuing alerts. Similarly, the final stage entails integrating the Android Speech-To-Text API for live transcription of phone calls. This involves configuring telephony services, capturing audio data, transcribing it using the Speech-To-Text API, and displaying the transcription. These stages adhere to the System Development Life Cycle (SDLC) methodology employing the Phased approach, ensuring a systematic and efficient approach to project development.

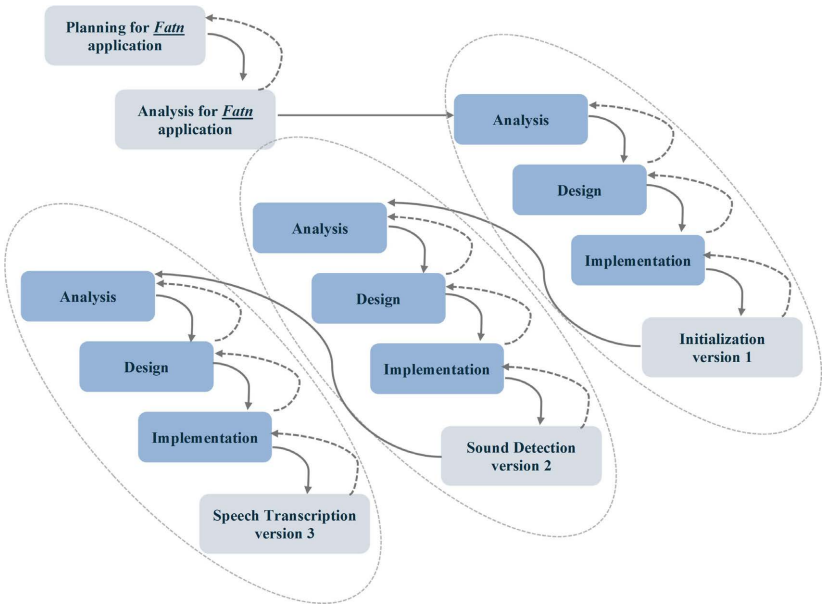


Figure 1. Phased development methodology for the application.

This project targets five critical alarm types: danger alarm, fire alarm, gas leak alarm, gunshot alarm, and tsunami alarm. These were selected to reflect both common hazards in Saudi Arabia and broader global threats, especially as they

relate to the needs of individuals who are deaf or hard of hearing.

- Fire and gas alarms are frequent risks in Saudi homes and workplaces due to climate conditions and gas usage.
- Danger alarms represent general threats requiring urgent action, such as evacuations or hazardous situations.
- Gunshot alarms, while rare in Saudi Arabia, are included to address possible security threats in public areas.
- Tsunami alarms are less common locally but are important for coastal regions and to support broader system adaptability.

The methodology is based on the principle that different alarms require varied levels of urgency and response. For deaf individuals, the inability to hear these alarms significantly increases risk and delays action. This highlights the need for inclusive alert systems that provide visual or tactile feedback.

Furthermore, in Saudi Arabia and the Arab world, there is a notable shortage of Arabic-based programs tailored for the hearing impaired. Therefore, this project aims to fill that gap by developing an alert system that improves safety and response time for this underserved community.

In addition, the dataset used in this study consists of 3719 samples, which is modest in size and lacks detailed metadata about source diversity, recording conditions, and background noise variations. Despite these limitations, it was selected as the most relevant and accessible dataset available for the specific alarm classes under investigation. To enhance the model's generalizability and address dataset constraints, several strategies were implemented:

- Data Augmentation: Techniques such as noise injection, pitch shifting, and time stretching were applied to artificially increase data variability and simulate diverse acoustic environments.
- Deep Neural Networks (DNNs): DNN architectures were employed to learn complex feature representations from limited data, improving detection accuracy.
- Multiple Instance Learning (MIL): MIL approaches enabled effective learning from weakly labeled or noisy data by considering bags of instances rather than relying on perfectly labeled individual samples.
- Feature Learning: Automated feature extraction techniques allowed the model to identify the most discriminative features, increasing adaptability across different acoustic conditions.

By integrating these methods, the approach overcomes the challenges of limited dataset size and variability, resulting in improved robustness and performance of the sound detection system.

4. Architectural Design

Architectural design entails the definition of the overall structure and organization of a system or software application. The two main architectural designs under discussion are real-time sound detection and speech recognition. This phase lays

the groundwork for the development process and offers direction for subsequent implementation and testing stages.

4.1. Designing an Architecture for Sound Detection Models

One of the central components of the application involves sound detection, that is needed to define the architectural design. The main steps for applying sound detection technology begin with data gathering and preprocessing. This is followed by feature extraction and model training. Subsequently, the application classifies the sound through real-time inference and then issues an alert to notify. The subsequent sections will elaborate on the fundamental procedures for constructing the model, and "Figure 2" illustrates the four main steps for the model-building process.

Figure 2. Key steps to implement sound detection model.

Data Collection and Preprocessing: To develop a classification sound model capable of identifying various emergency sounds, the initial step involves gathering datasets suitable for model construction. Accordingly, the app has amassed datasets for different classes such as danger alarm, fire alarm, gas alarm, gunshot, and tsunami alarm. Each class dataset aims to meet specific criteria aimed at enhancing accuracy in sound detection and classification. This entails acquiring sufficiently extensive datasets to facilitate effective model training, while also ensuring a diverse range of scenarios and conditions are encompassed within the dataset. "Table 1" outlines each sound class, along with the number of samples and their corresponding references.

Table 1. Datasets for sound detection model.

Reference	Classes	Number of samples
[16] [17]	Danger alarm	820
[16] [17]	Fire alarm	868
[17]	Gas alarm	828
[18]-[20]	Gunshot	369
[20]	Tsunami alarm	834

[a]Datasets for sound detection model.

After collecting the data, preprocessing is crucial. This involves manually removing redundancy and inconsistencies in WAV data. The data is further segmented into one-second intervals to improve input data quality and increase the training data size. These steps collectively enhance overall quality and ensure a consistent format across all classes. Resampling is applied to each segment to

maintain a uniform sample rate of 16 kHz. "Table 2" illustrates the changes in the number of sample data resulting from these preprocessing techniques.

Table 2. Datasets for sound detection model after segmenting.

Reference	Classes	Number of samples	After prepressing
[16] [17]	Danger alarm	820	977
[16] [17]	Fire alarm	868	1166
[17]	Gas alarm	828	905
[18]-[20]	Gunshot	369	788
[20]	Tsunami alarm	834	960

[b]Datasets for sound detection model after segmenting.

In spite of the small data approximately for the model, it back to rare to find the type of sound with different sources or background to enhance the training.

Feature Extraction: After collecting and preparing data, the audio information must undergo a transformation to extract features suitable for input into the model. In the application, data is prepared for a sound detection model using the YAMNet model to enhance the quality of sound classification.

YAMNet, designed for audio event classification, identifies 521 audio events and provides per-frame predicted scores, embeddings, and log mel spectrograms. The embeddings, capturing average-pooled features, are instrumental in building specialized audio classifiers without extensive labeled data. YAMNet-generated embeddings find applications in various audio processing and machine learning tasks, serving as input features for models like audio classification. However, YAMNet's classifier outputs require calibration for accurate per-class score thresholds [3] [9]. In the app, these calibrated embeddings from YAMNet are used to extract features for emergence alarms, offering condensed and informative representations of audio waveforms, including spectral characteristics, temporal patterns, and semantic features.

Model Training: After preprocessing and converting the data into embeddings via YAMNet, the subsequent stage involves training the model. Prior to this, the dataset is partitioned into training, validation, and test subsets, with allocations of 60%, 20%, and 20% respectively, utilized for training and validating the model. A Deep Neural Network (DNN) is chosen for sound detection because of its proven accuracy. The model content four layers includes Audio Input Layer, YAMNet Layer, Model Architecture, and Classifier Layer.

1) *Audio Input Layer*: The audio dataset is loaded from WAV files and preprocessed to ensure uniformity in sample rate and channel format. The input layer receives the audio data as a tensor.

2) *YAMNet Layer*: YAMNet is a pre-trained model for audio analysis, specifi-

cally designed to extract features from audio data. In this design, YAMNet is utilized as a Keras layer to process the audio embeddings.

3) *Model Architecture*: Model architecture is defined as a Sequential model in Keras. It consists of layers for processing the audio embeddings extracted by YAMNet. These layers include Dense layers for feature transformation and dimensionality reduction.

4) *Classifier Layer*: The output of the model is passed through a classifier layer, such as ReduceMeanLayer, which computes the mean of the output embeddings along a specified axis. This step helps in aggregating information from the embeddings to make predictions.

In other words, the model undergoes training with a labeled dataset. This dataset comprises audio embeddings obtained through YAMNet. The input consists of these embeddings, while the output includes various labels like danger alarm, fire alarm, gas alarm, gunshot, and tsunami alarm. Refer to "Figure 3" for an illustration of this training process.

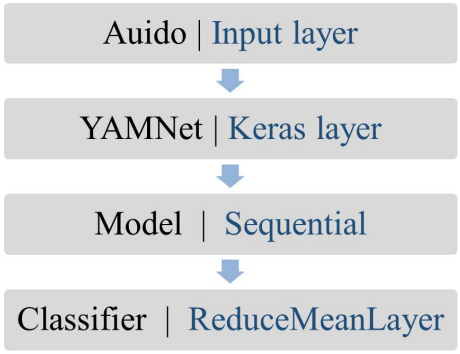


Figure 3. The architecture design for DNN classification model.

Evaluation and validation: The final stage of model development involves assessing its quality through evaluation and validation. Following training on the training dataset and validation on a separate dataset, the model's performance is assessed using a test dataset. This phase entails making predictions on the test set with the trained model and evaluating its performance using metrics such as precision, recall, and F1-score. A classification report is generated to analyze these metrics for each class, and a confusion matrix visualization provides further insights. Plotting the training and validation loss over epochs aids in understanding the model's learning progression.

In conclusion, the Architectural Design section outlines the system's structure and organization for real-time sound detection and speech recognition. It covers key steps such as data collection, preprocessing, feature extraction with YAMNet, model training using a Deep Neural Network (DNN), and performance evaluation. "Figure 4" provides an overview of these steps, with adaptations made to improve clarity.

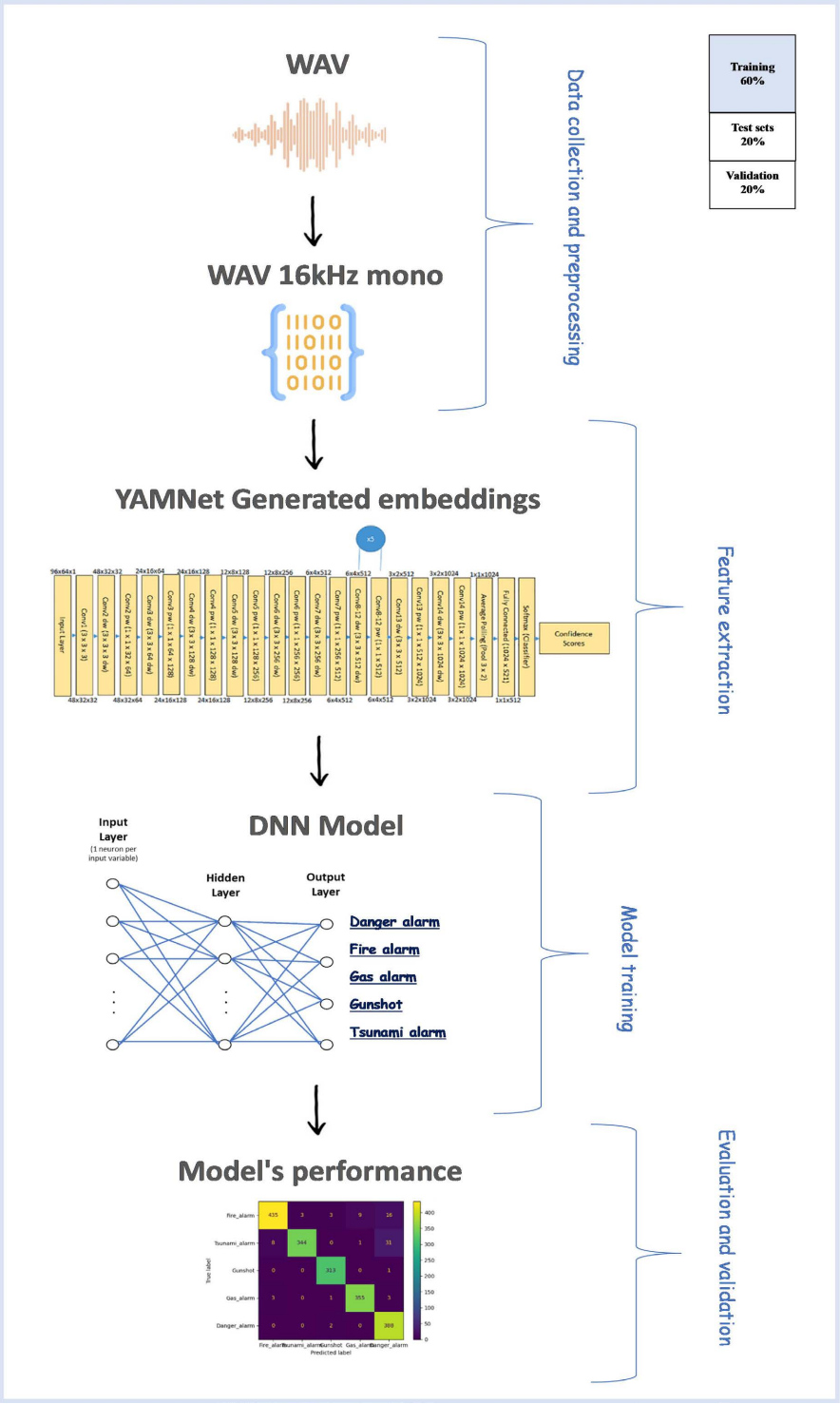


Figure 4. The comprehensive approach for the sound detection model.

4.2. Designing an Architecture for Speech Recognition

The second phase of the app involves integrating speech recognition technology to provide real-time transcripts during calls to authorities. One popular approach to implementing this technology is utilizing the Android speech-to-text API. This

API streamlines the integration of speech recognition functionality into Android applications. At its core, the API functions through a series of components and processes aimed at capturing audio input, analyzing it for speech content, and producing corresponding text output [21] [22].

"Figure 5" illustrates the primary steps of the Android Speech-To-Text API. The BroadcastReceiver serves as a crucial component in the implementation of the Android Speech-to-Text API for speech recognition. When a user interacts with the app's speech recognition feature, the BroadcastReceiver captures the relevant intents or broadcast messages sent by the Android system or other components within the app. These intents typically signal the start or end of speech recognition processes. Upon receiving the appropriate intent, the BroadcastReceiver triggers the Speech-to-Text API, which initiates the speech recognition process. This API works with the device's microphone to record the user's speech input as it happens. After the user stops speaking, the Speech-to-Text API analyzes the audio data, changes it into text, and then sends the recognized text back to the app.

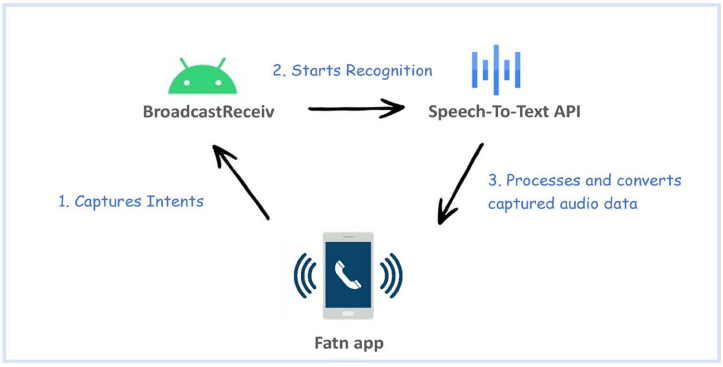


Figure 5. Model architecture for speech recognition technology.

5. Experiment and Implementation

After finalizing the architectural design for the proposed system, the next step involves translating this design into actual code. To initiate the implementation of the app, the process can be broken down into three main steps aimed at encompassing its core functionality. Firstly, it is imperative to elucidate the Android Studio environment, along with the necessary libraries and requirements essential for the app. Secondly, the app must be equipped to recognize emergency sounds and promptly alert the user on their Android device. This entails the implementation and construction of a sound detection model to seamlessly integrate with the Android device. Finally, a comprehensive explanation is required on the integration of the Speech-To-Text API with Android Studio to facilitate real-time transcription during calls.

5.1. Configuration Environment with Android Studio

The "Configuring the Environment with Android Studio" section explains setting

up the app development environment using Android Studio. This involves configuring settings, tools, and dependencies to enable building, testing, and deploying the app on Android devices. The app targets Android SDK versions 21 (Android 5.0) to 33 (Android 12) to ensure compatibility across a wide range of devices.

The configuration includes three main components:

1) *Account Management:* Covers user registration, login, and profile editing.

2) *Sound Detection:* Integrates a DNN model using TensorFlow Lite for sound detection and user notifications.

3) *Speech Recognition:* Enables API connectivity for transcribing speech to text and initiating emergency calls with the Saudi Authority.

The app utilizes Java 8 core library desugaring for modern feature support and includes dependencies such as Firebase, AndroidX components, TensorFlow Lite, and gRPC. These facilitate machine learning, cloud services, UI components, and network communication. Its interactive interface allows user registration and login through Firebase, sound detection of alarms using DNN, and emergency API connectivity via gRPC. Detailed implementation is discussed in subsequent sections.

5.2. Sound Detection Model Implementation

To implement a DNN model for classifying five sound categories—danger alarm, fire alarm, gas alarm, gunshot, and tsunami alarm—various essential Python libraries and modules are utilized within a Jupyter Notebook environment. Key dependencies include os, IPython.display, matplotlib.pyplot, numpy, pandas, tensorflow, tensorflow_hub, tensorflow_io, Librosa, soundfile, keras.models, sklearn.metrics, and tqdm. Some code segments were adapted from online sources [3] [23] to aid in the app's development.

Each stage of the model's implementation leverages specific functions from these libraries, aligning with the architectural design for sound detection. To simplify understanding, these functions and their roles are organized by implementation stages and summarized in "Table 3" for clarity and efficiency.

Table 3. Datasets for sound detection model after segmenting.

Stages	Functions and methods
Data collection and preprocessing	- segment_wav - load_wav_16k_mono
Feature extraction	- download_yamnet_model - tf.data.Dataset.from_tensor_slices - load_wav_for_map and map - extract_embedding - unbatch
Model training	- split_dataset - compile_model - fit

Continued

Evaluation and validation	- evaluate - predict - classification_report - ConfusionMatrixDisplay - plt.plot
Integrated with Android Studio Environment	- TFLiteConverter

[a]Datasets for sound detection model after segmenting.

Data Collection and Preprocessing: The process comprises collecting audio data from diverse origins and categorizing it into five groups. To enhance the data quality, redundancies and discrepancies in WAV files are eliminated manually. Additionally, the WAV files are segmented using the segment_wav function. This function divides the total duration of the WAV file into 1-second segments, adjusts shorter segments to ensure they are all 1 second long, and saves each segment as a new WAV file in a designated output folder.

To interact with the data effectively, an Excel sheet is needed to map the label classes. This sheet consists of 4796 rows and five columns: "n" indicating the list number of sounds, "target" representing label classes numerically (0: danger alarm, 1: fire alarm, 2: gas alarm, 3: gunshot, 4: tsunami alarm), "category" showing label classes, 'filename' indicating the path for the sound, and 'fold' for sound classification (ranging from 1 to 5).

The load_wav_16k_mono function is responsible for loading WAV files, decoding them, converting them to float tensors, squeezing them to remove unnecessary dimensions, and resampling them to 16 kHz single-channel audio using TensorFlow operations. This function performs all necessary steps to prepare the audio data for further processing, ensuring it is properly formatted and ready for feature extraction or any subsequent analysis.

Feature Extraction: To extract features from a WAV file using the YAMNet model, it is necessary to adapt the dataset into a suitable format. Firstly, the YAMNet model needs to be downloaded. As discussed earlier, YAMNet, as a pretrained model designed for audio analysis tasks, is equipped to produce high-level features or embeddings that encapsulate the semantic information embedded within the input audio.

Once the YAMNet model has been successfully loaded using the download_yamnet_model function, the subsequent step involves preparing the data for feature extraction by creating the primary dataset. This dataset is established utilizing TensorFlow's tf.data.Dataset.from_tensor_slices function. Each element within this dataset encompasses details pertaining to the filenames of audio files, their corresponding labels, and folds.

To efficiently handle the audio data in this dataset, it undergoes a sequence of transformations. Initially, the dataset is passed through the load_wav_for_map function using the map function. This process involves reading and preprocessing

the audio files to ensure they are properly formatted for further feature extraction.

After preparing the audio data, it undergoes transformation with the extract_embedding function, leveraging the YAMNet model to obtain embeddings. Then, the dataset is unbatched with the unbatch function to enable processing of individual elements, each now including embeddings, labels, and fold details. This approach optimizes the utilization of extracted features for tasks such as training audio classification models.

Model Training: During the DNN model's training phase, the split_dataset function plays a crucial role. This function divides a dataset into training, validation, and test subsets using specified fold values, grouping data into categories of less than 3, 4, and 5. These subsets are allocated 60% for training, 20% for validation, and 20% for testing, ensuring a balanced distribution of sound classes. The function enhances performance by caching the dataset, filtering it based on fold values, removing the fold column, and adjusting the dataset accordingly. Additionally, it improves training efficiency by shuffling, batching, and prefetching each subset.

The initiation with the creation of a DNN model in TensorFlow's Keras API. Initially, the model structure is defined, encompassing layers for audio input, integration of YAMNet for feature extraction, additional dense layers for further feature extraction, and a final dense layer for classification. Subsequently, the model is compiled for training, specifying parameters such as the loss function, optimizer, and evaluation metrics. Training commences with the fit method, implementing early stopping to prevent overfitting, and continues for a set number of epochs.

After training, YAMNet integration improves the model's ability to analyze audio, especially for Android Studio apps. YAMNet is incorporated as a distinct Keras layer, functioning independently from the model's architecture. This setup treats YAMNet as a fixed feature extractor, generating embeddings from audio inputs, depicted in "Figure 6". The trained DNN model subsequently utilizes these embeddings to predict outcomes for input audio segments.

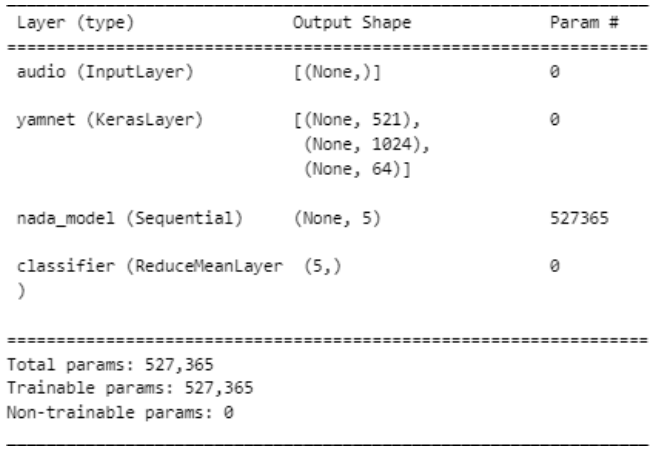


Figure 6. YAMNet is integrated as a Keras layer.

Evaluation and Validation: Following the training of the DNN, its performance is assessed on the test dataset using the evaluate function, yielding metrics like loss and accuracy. The loss value signifies the model's prediction error rate, while accuracy represents the percentage of correctly classified samples. The evaluation reveals a loss of 0.1863 and an accuracy of 95.77%, indicating the DNN's adept classification of audio samples in the test dataset.

Furthermore, the trained model generates predictions on the test dataset using the predict function. These predictions are utilized to calculate metrics like precision, recall, and F1-score for each class through the classification_report function. This report offers thorough insights into the model's performance for individual classes, as illustrated in "Figure 7".

```
60/60 [==============================] - 0s 2ms/step
               precision    recall  f1-score   support

   Fire_alarm       0.98      0.93      0.95       466
Tsunami_alarm       0.99      0.90      0.94       384
      Gunshot       0.98      1.00      0.99       314
   Gas_alarm        0.97      0.98      0.98       362
 Danger_alarm       0.88      0.99      0.94       390

     accuracy                           0.96      1916
    macro avg       0.96      0.96      0.96      1916
 weighted avg       0.96      0.96      0.96      1916
```

Figure 7. Segment_wav function for evaluation and validation.

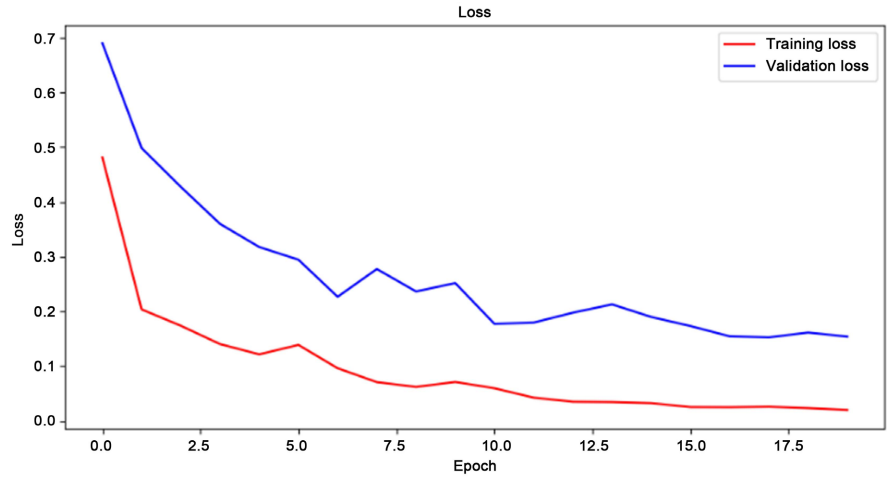
The ConfusionMatrixDisplay is employed to visually represent the confusion matrix, depicted in "Figure 8". This matrix offers insights into the model's classification performance, displaying metrics like true positives, false positives, true negatives, and false negatives for each class. It helps assess the model's strengths and weaknesses in classification tasks.

The training and validation loss curves are graphed using plt.plot to visualize the model's learning progress across epochs, aiding in detecting overfitting or underfitting. Likewise, accuracy curves are plotted to monitor accuracy improvements. These visualizations in "Figure 9" offer insights into the model's training dynamics and generalization abilities.

Integrated with Android Studio Environment: The final stage of implementing the sound detection model in the app involves configuring it with an Android device. This step is crucial for deploying machine learning models on mobile phones with constrained resources. The process includes converting the DNN model to TensorFlow Lite format using TFLiteConverter and saving it to a specified file path. Once converted, the model is ready to be used in Android Studio to connect with the Android device and integrate into the app.

The ClassificationActivity class in Andriod Studio, is responsible and serves as the core component for real-time sound detection within the app, providing users with immediate feedback upon detecting predefined sound events. It utilizes TensorFlow Lite for inference on sound data captured via the device's microphone.

Below is a summary of the key functionalities within the class:



[ ]

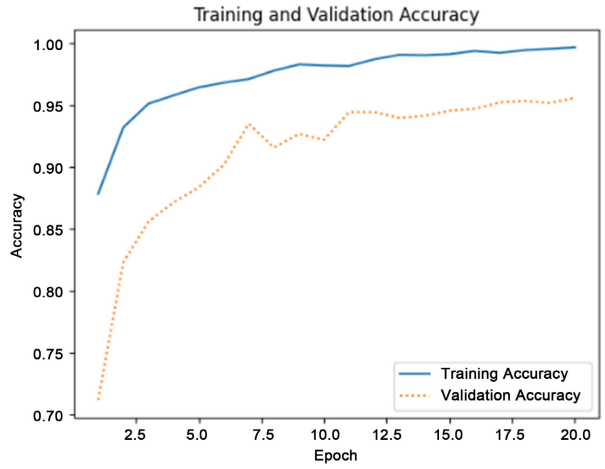


Figure 8. ConfusionMatrixDisplay function for representing the confusion matrix.

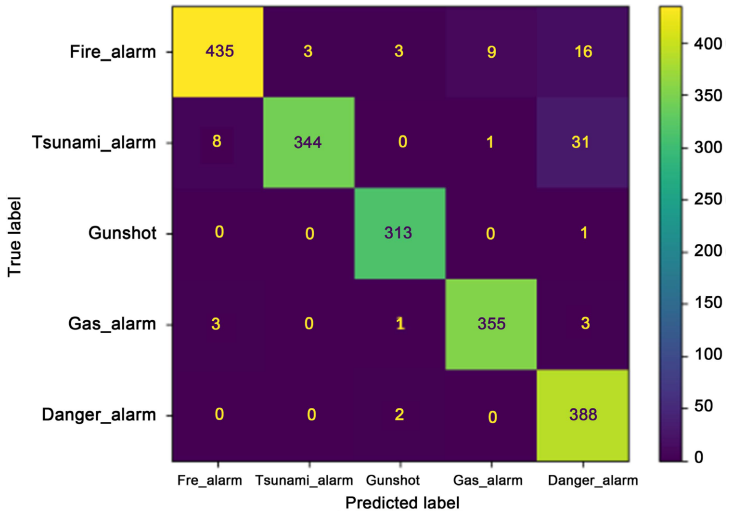


Figure 9. plt.plot function for monitor accuracy improvements.

1) *Model Initialization:* The class loads the TensorFlow Lite model (SDModel.tflite) in the onCreate method, preparing it for inference.

2) *Audio Recording:* The startRecording method initializes audio recording using the device's microphone. Sound data is continuously captured, processed, and classified.

3) *Feature Extraction:* The convertToArray and convertToInputFormat methods preprocess the raw audio data into the format expected by the TensorFlow Lite model.

4) *Model Inference:* The classifySound method runs inference on the preprocessed audio data using the loaded TensorFlow Lite model, determining the sound category.

5) *Notification and UI Update:* The app sends a notification and changes its interface when it identifies a sound above a certain confidence level (THRESHOLD).

6) *Permission Handling:* The app requests permission to record audio (Manifest.permission.RECORD_AUDIO) if not already granted.

5.3. Speech Recognition Implementation

In this section of the app, the focus lies on integrating speech recognition functionalities with the Speech-To-Text API within Android Studio. This integration aims to enable real-time transcription during phone calls. There are three classes associated with this aspect:

The first class, ActivityNumber, is concerned with selecting the appropriate authority number in Saudi Arabia. The users can contact various authorities, such as civil defense, police, and emergency services in case of any emergency.

Following that, the OutgoingBroadcastReceiver class initializes the call and allows users to commence transcription of the call. The BroadcastReceiver is tasked with this function.

The SpeechRecognitionActivity class, pivotal to speech recognition within the app, utilizes the Android Speech-To-Text API. Upon creation, the class initializes a SpeechRecognizer instance and configures it to recognize speech input in Arabic. As the user speaks, recognition results are continuously received and processed. These results are displayed in a RecyclerView to provide real-time feedback. The startSpeechRecognition method initiates speech recognition by creating and starting a RecognizerIntent with appropriate parameters. Additionally, the class handles the onDestroy event to release resources properly. Overall, this class forms a crucial component for enabling speech-to-text functionality within the app, enhancing user interaction and accessibility.

6. Evaluation and Testing

The Fatn app underwent an initial usability evaluation after the requirements gathering and analysis phase, employing a prototype model tested by 10 users through the Figma platform [24]. This testing phase collected qualitative feedback

that led to key improvements such as the addition of visual emergency alerts (a red flash bar and vibration) and an enhanced, more appealing app icon. Usability evaluation followed ISO standards, which define usability as the user's ability to achieve specific goals within a given context, measured across dimensions including effectiveness, efficiency, satisfaction, learnability, and memorability [25]. Fatn's usability was assessed via pre- and post-tests, showing a strongly positive user response and high satisfaction levels in areas like accessibility, task completion, navigation, and overall usability, as detailed in the Fatn usability scale **Table 4**.

Table 4. Fatn usbility scale.

Usability scale questions	Scale points				
	1	2	3	4	5
Do you find Fatn accessible for users with diverse abilities?					10
How efficiently do you accomplish tasks using Fatn?					10
Do you find the design of Fatn visually appealing?			1	1	8
Overall, how satisfied are you with the usability of Fatn?					10
Are Fatn's features easy to discover and access?					10
Does it provide visual alerts for important events		1	3		6
How helpful is the captioning content for deaf or hard of hearing users?				1	9

Test cases focused on verifying software components demonstrated that all features functioned as intended and contributed positively to the user experience. The app's key functionalities include user account creation, login, password reset, emergency sound detection with categorized alerts, and the ability to call authorities and transcribe conversations to aid communication.

However, the evaluation so far lacks detailed documentation of real-world testing conditions, including the number and demographic details of participants, especially those from the target deaf and hard-of-hearing communities, and the variety of environmental contexts in which testing was conducted. The testing primarily took place in controlled or simulated environments without extensive field trials. To fully assess the practical utility and reliability of Fatn, a more comprehensive real-world evaluation is essential. This should involve a larger, more diverse group of deaf and hard-of-hearing users engaging with the app in varied real-life settings to validate its effectiveness, user experience, and robustness under different conditions.

7. Result and Discussion

The proposed sound detection system demonstrates robust performance, achieving an overall accuracy of 96% and a low loss value of 0.1863, indicating minimal prediction errors. Evaluation metrics, such as precision, recall, and F1-score (as presented in Figure 7, confirm consistent classification capabilities across five sound categories: Danger Alarm, Fire Alarm, Gas Alarm, Gunshot, and Tsunami Alarm. Notably, the Fire Alarm and Gas Alarm classes achieved F1-scores of 0.95 and 0.98, respectively, reflecting the model's ability to classify critical emergency sounds accurately. However, the Danger Alarm class recorded a slightly lower F1-score of 0.94 and precision of 0.88, suggesting challenges in distinguishing acoustically overlapping sounds.

The confusion matrix in Figure 8 provides further insights, revealing minimal misclassifications, such as 8 Fire Alarm sounds being misclassified as Tsunami Alarms and 3 vice versa, as well as some confusion between Danger Alarm, Gas Alarm, and Fire Alarm. These misclassifications are likely due to similarities in their acoustic features. This pattern indicates a potential bias in class boundaries, especially for alarms with overlapping frequency profiles or occurring under similar background conditions. These findings highlight opportunities to refine feature extraction techniques and expand the dataset to better address edge cases.

Training and validation curves for accuracy and loss Figure 9 validate the model's reliability, with training accuracy stabilizing above 95% and validation accuracy closely following, demonstrating minimal overfitting and effective generalization. The final model shows a high capacity for learning meaningful representations from a relatively modest dataset (3719 samples), further supported by data augmentation and regularization techniques.

To contextualize the reported performance, we considered traditional baseline approaches such as MFCC feature extraction with Support Vector Machines (SVMs) and direct classification using YAMNet softmax outputs. While these were not implemented in the current version, previous studies provide reference benchmarks. For example, Liu *et al.* [26] used a DNN with MFCC features to classify cough sounds and achieved 90.1% accuracy in identifying positive cases and 85% for negative cases. Similarly, Amoh and Odame [27] reported 86.8% and 92.7% accuracy for positive and negative cases, respectively. These findings suggest that traditional MFCC-based classifiers typically achieve accuracy ranges of 85% - 90%. As such, our model's 95.77% accuracy represents a significant improvement, demonstrating the benefits of using deep learning, data augmentation, and Multiple Instance Learning (MIL) for more effective sound classification.

Integration with TensorFlow Lite enables real-time sound detection on resource-constrained Android devices, making the system practical for deployment. Real-world testing confirmed the app's ability to provide timely emergency notifications. Additionally, the incorporation of Speech-To-Text functionality for Ar-

abic speech transcription during calls further enhances accessibility, particularly in emergency scenarios.

Compared to traditional assistive technologies, the system introduces significant advancements through machine learning-based sound detection and real-time transcription functionalities. Its modular architecture, leveraging YAMNet embeddings for feature extraction, ensures scalability, efficient Android integration, and broad compatibility.

Despite its promising results, the system faces challenges that warrant further refinement. Expanding the dataset to include more diverse sound samples and background noise scenarios could enhance performance in real-world environments. Optimizing the TensorFlow Lite model is also essential to improve efficiency on mobile devices. Furthermore, speech recognition performance could be improved by incorporating advanced noise reduction techniques to address issues with overlapping conversations and noisy environments.

The proposed system has significant implications for individuals with hearing impairments, enhancing safety and communication by providing real-time sound detection, emergency notifications, and transcription features. Future developments could include expanding support for multiple languages, integrating location-based alerts, and leveraging IoT connectivity for broader applications.

In conclusion, the results affirm the feasibility and effectiveness of the system in addressing critical challenges for individuals with hearing impairments. While limitations exist, the system sets a strong foundation for future research and development, paving the way for impactful advancements in assistive technologies.

8. Conclusions

The artificial intelligence center created this project to address safety and communication problems for the deaf and hard-of-hearing members of society in Saudi Arabia. Combined with real-time sound detection and speech recognition technology, this project seeks to offer alerts for danger, instant communication, and further independence for individuals with hearing deficits. This initiative is a big step toward creating a more secure and inclusive community in line with worldwide accessibility improvements.

The system's design draws from a comprehensive literature review, highlighting advancements in machine learning—particularly sound event detection—and their applications in addressing safety and communication needs. The development process employs tools such as Android Studio, TensorFlow Lite, and Speech-to-Text APIs, ensuring compatibility with Android devices. Key functionalities include real-time sound detection, emergency alert generation, and speech-to-text capabilities.

The Architectural Design section emphasizes system structure and organization, detailing steps such as data collection, preprocessing, feature extraction with

YAMNet, and model evaluation.

The core model design and training process exemplify the project's technical foundation. Audio data, preprocessed and converted into embeddings via YAMNet, is used to train a Deep Neural Network (DNN) model. The architecture includes four layers: Audio Input Layer, YAMNet Layer, Model Architecture, and Classifier Layer.

The dataset is split into training, validation, and testing subsets (60%, 20%, 20%) to ensure robust evaluation. The trained DNN achieves strong performance, with a loss of 0.1863 and accuracy of 95.77%, supported by precision, recall, F1-score, and confusion matrix metrics. Training and validation curves further illustrate the model's ability to generalize effectively.

Looking ahead, future enhancements aim to expand the application's capabilities and user impact. These include introducing customizable alerts, flashing notifications, and integration with government authorities. Features like sensor-based hazard detection, sign language support, location tracking, and video call functionality are proposed to further enhance safety and communication. Additionally, the app will explore detecting a broader range of hazards, such as car horns and alarms, and improving sound recognition accuracy. By addressing these areas, the project seeks to continually evolve, offering a comprehensive solution that empowers the deaf and hard-of-hearing community while advancing societal inclusivity.

Conflicts of Interest

The authors declare no conflicts of interest regarding the publication of this paper.

References

[1] World Health Organization (2023) Deafness and Hearing Loss. https://www.who.int/news-room/fact-sheets/detail/deafness-and-hearing-loss

[2] Abdallah, E.E. and Fayyoumi, E. (2016) Assistive Technology for Deaf People Based on Android Platform. *Procedia Computer Science*, **94**, 295-301. https://doi.org/10.1016/j.procs.2016.08.044

[3] GOOGLE (2020) Yamnet. Kaggle. https://www.kaggle.com/models/google/yamnet

[4] Alkhalifa, S. and Al-Razgan, M. (2018) ENSSAT: Wearable Technology Application for the Deaf and Hard of Hearing. *Multimedia Tools and Applications*, **77**, 22007-22031. https://doi.org/10.1007/s11042-018-5860-5

[5] Ozarkar, S., Chetwani, R., Devare, S., Haryani, S. and Giri, N. (2020) AI for Accessibility: Virtual Assistant for Hearing Impaired. 2020 11*th International Conference on Computing, Communication and Networking Technologies* (*ICCCNT*), Kharagpur, 1-3 July 2020, 1-7. https://doi.org/10.1109/icccnt49239.2020.9225392

[6] Saifan, R.R., Dweik, W. and Abdel-Majeed, M. (2018) A Machine Learning Based Deaf Assistance Digital System. *Computer Applications in Engineering Education*, **26**, 1008-1019. https://doi.org/10.1002/cae.21952

[7] Bragg, D., Huynh, N. and Ladner, R.E. (2016) A Personalizable Mobile Sound Detec-

tor App Design for Deaf and Hard-of-Hearing Users. *Proceedings of the* 18*th International ACM SIGACCESS Conference on Computers and Accessibility*, Reno, 23-26 October 2016, 3-13. https://doi.org/10.1145/2982142.2982171

[8] Chen, L., Tsai, C., Chang, W., Cheng, Y. and Li, K.S. (2016) A Real-Time Mobile Emergency Assistance System for Helping Deaf-Mute People/Elderly Singletons. 2016 *IEEE International Conference on Consumer Electronics* (*ICCE*), Las Vegas, 7-11 January 2016, 45-46. https://doi.org/10.1109/icce.2016.7430516

[9] Valliappan, N.H., Pande, S.D. and Reddy Vinta, S. (2024) Enhancing Gun Detection with Transfer Learning and Yamnet Audio Classification. *IEEE Access*, **12**, 58940-58949. https://doi.org/10.1109/access.2024.3392649

[10] Sound Alert. Alerts to Sounds around You (No Date). Convert Sounds into Visual and Sensory Notifications. http://www.soundalert.co/

[11] APKPure.com (2015) The Deaf and Hearing Impaired APK for Android Download. https://apkpure.com/the-deaf-and-hearing-impaired/kr.ac.kaist.isilab.doyeob.humanityProject2

[12] Google. Live Transcribe and Notification Apps on Google Play (No Date). https://play.google.com/store/apps/details?id=com.google.audio.hearing.visualization.accessibility.scribe

[13] Rogervoice. Caption All Your Phone Calls Instantly (No Date). https://rogervoice.com/en/

[14] Google. Kamnapp كلنا أمن - التطبيقات على Google Play (No Date). https://play.google.com/store/apps/details?id=sa.gov.moi.securityinform&hl=ar

[15] Home (2023) TapSOS. https://tapsos.com/

[16] Gorgolewski, C. (2020) UrbanSound8K, Kaggle. https://www.kaggle.com/datasets/chrisfilo/urbansound8k

[17] Desnug, D. (2022) Audio-Alarm-Dataset. Kaggle. https://www.kaggle.com/datasets/devisdesnug/audio-alarm-dataset

[18] Nugroho, D. (2022) RNN-MFCC. Kaggle. https://www.kaggle.com/code/devisdesnug/rnn-mfcc/input

[19] Aydemir, E. (2021) Gunshot Audio Dataset. Kaggle. https://www.kaggle.com/datasets/emrahaydemr/gunshot-audio-dataset

[20] Free SFX. Free Sound Effects (No Date). https://freesfx.co.uk/

[21] Developex Blog (2017) Overview of Speech Recognition APIs for Android Platform. https://developex.com/blog/overview-of-speech-recognition-apis-for-android-platform/

[22] Android (2023) Speech: Android Developers. https://developer.android.com/reference/android/speech/package-summary

[23] Rathore, S. (2019) RSHIVAM08 Overview, GitHub. https://github.com/rshivam08

[24] Figma (2024) Figma Community: Explore Templates, Plugins, and Widgets Published by the Community (No Date). https://www.figma.com/community

[25] Ntoa, S. (2025) Usability and User Experience Evaluation in Intelligent Environments: A Review and Reappraisal. *International Journal of Human–Computer Interaction*, 41, 2829-2858. https://doi.org/10.1080/10447318.2024.2394724

[26] Liu, J.M., You, M.Y., Wang, Z., Li, G.Z., Xu, X.H. and Qiu, Z.M. (2014) Cough De-

tection Using Deep Neural Networks. 2014 *IEEE International Conference on Bioinformatics and Biomedicine* (*BIBM*), Belfast, 2-5 November 2014, 560-563. https://doi.org/10.1109/bibm.2014.6999220

[27] Amoh, J. and Odame, K. (2016) Deep Neural Networks for Identifying Cough Sounds. *IEEE Transactions on Biomedical Circuits and Systems*, **10**, 1003-1011. https://doi.org/10.1109/tbcas.2016.2598794