iJRASET
International Journal For Research in Applied Science and Engineering Technology



INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 13 **Issue:** XI **Month of publication:** November 2025

DOI: https://doi.org/10.22214/ijraset.2025.75260

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 13 Issue XI Nov 2025- Available at www.ijraset.com

Keyword Spotting System

Lakshmi B[1], Nathiya M[2], Navyaa S[3], Mrs. Meena V[4]
[1, 2]*Bachelor of Engineering in Computer Science and Engineering, Adhiyamaan College of Engineering, ANNA University, Chennai*
[3]*M.E., Assistant Professor, Department of Computer Science and Engineering, Adhiyamaan College of Engineering, ANNA University, Chennai*

Abstract: ***The project "KEYWORD SPOT" is a Machine Learning-based Web Application designed for Few-Shot Language Agnostic Keyword Spotting (FSLAKWS), enabling accurate detection of spoken keywords across multiple languages using minimal training data. By leveraging Few- Shot Learning with Prototypical Networks, the system achieves rapid keyword recognition without extensive datasets or frequent retraining. It integrates Phonetic Modeling for multilingual versatility and utilizes pre-trained models with optimized pre-processing techniques to ensure low latency and real-time performance. Developed using Python Flask for backend processing, the application offers a user-friendly interface and efficient performance. Audio features are processed using Librosa and FFmpeg, while MinIO provides secure data management. With support for continuous learning, the system adapts to new keywords over time, enhancing accuracy and scalability. Overall, KEYWORD SPOT delivers a lightweight, adaptive, and cost- effective solution for voice-driven systems such as virtual assistants, chatbots, and smart devices, contributing to the advancement of multilingual and intelligent speech recognition technologies.***

Keywords: Few-Shot Learning, Keyword Spotting, Speech Recognition, Phonetic Modeling, Log-Mel Spectrogram, Librosa, TensorFlow, Flask, Real-Time Processing, Multilingual Detection.

I. INTRODUCTION

A. Overview

In the modern era of Artificial Intelligence (AI), voice-driven technology has become one of the most important innovations shaping human-computer interaction. From virtual assistants such as Amazon Alexa, Google Assistant, and Apple Siri to call automation systems, speech interfaces are redefining how people communicate with digital devices.The ability of a system to accurately detect specific keywords or commands from a continuous stream of speech is known as Keyword Spotting (KWS)**.** It is a fundamental component of many applications that depend on voice commands**,** wake-word detection**,** audio-based search**,** and contextual interaction**.** However, most traditional keyword spotting systems rely on large-scale supervised training. These systems require massive datasets of labeled audio recordings to perform well. Collecting and labeling such data for multiple languages is expensive and time-consuming. Furthermore, every time a new keyword is introduced, the system must undergo complete retraining, resulting in inefficiency and scalability issues. To address these limitations, the proposed project titled "KEYWORD SPOT – Few-Shot Language Agnostic Keyword Spotting System" presents an innovative approach that leverages Few-Shot Learning (FSL) and Phonetic Modeling to enable efficient and language-independent keyword detection. The system is capable of recognizing new keywords from only a few training examples, thereby reducing the dependency on massive datasets. It also integrates pre-trained deep learning models to accelerate the recognition process and achieve low-latency, real-time performance.

KEYWORD SPOT is implemented as a web-based application using Python Flask for backend processing and HTML5, CSS3, and JavaScript for frontend development. The system uses Librosa and FFmpeg for feature extraction and audio preprocessing, while MinIO serves as the data storage system for managing large audio files. The project's modular design, scalability, and multilingual adaptability make it suitable for a wide range of practical applications in voice assistants**,** automated transcription systems**,** customer service**,** and human-computer interaction platforms**.**

B. Background of the Study

Speech is the most natural form of communication between humans. The goal of speech technology is to replicate this natural communication between humans and machines. In recent years, the advancements in deep learning and computational linguistics have significantly improved the accuracy of speech recognition systems. Yet, despite these advancements, multilingual and low-resource keyword detection remains a challenging task. Traditional keyword spotting systems rely heavily on Automatic Speech Recognition (ASR) models that transcribe entire audio sequences into text before analyzing them.

©IJRASET: All Rights are Reserved

This process is computationally expensive and prone to errors, especially in noisy environments. To make keyword detection more efficient, researchers have explored direct keyword spotting techniques that bypass full transcription. These systems classify short speech segments into predefined keyword categories. The concept of Few-Shot Learning (FSL) introduces a paradigm shift in this domain. Unlike conventional models that need thousands of examples per class, Few-Shot Learning models can learn new tasks or classes from only a handful of samples. This approach is particularly useful for voice- based systems, where collecting large multilingual datasets is impractical. By integrating Prototypical Networks, the proposed system can create vector representations of each keyword and compare new inputs based on similarity in feature space. This enables rapid learning and flexible adaptation to new languages and accents.

C. Problem Definition

In existing keyword spotting systems, the model is trained using a fixed vocabulary and a large dataset. When new keywords are introduced or when users from different linguistic backgrounds use the system, its performance degrades. The model also struggles to generalize across multiple languages due to phonetic variations. Moreover, training such systems from scratch for every new keyword set is computationally expensive.

Hence, there is a need for a system that can:

Recognize new keywords efficiently using few examples.

Operate effectively across multiple languages and accents. Maintain low-latency performance for real-time interaction. Reduce computational overhead and training cost.

The KEYWORD SPOT project aims to resolve these issues by introducing a Few-Shot Learning- based Language Agnostic Keyword Spotting System capable of adaptive learning and cross-lingual generalization.

D. Objectives of the Project

The main objectives of the KEYWORD SPOT system are as follows:

To develop a machine learning-based web application that performs multilingual keyword recognition.

To implement Few-Shot Learning using Prototypical Networks, allowing recognition of new keywords with minimal data.

To integrate Phonetic Modeling for enhanced multilingual and accent-independent performance. To ensure low latency and high accuracy, enabling real-time keyword detection.

To build a scalable and user-friendly web interface for audio input and keyword visualization.

To support continuous learning, allowing the system to improve over time without retraining from scratch.

E. Scope of the Project

The KEYWORD SPOT system focuses on delivering an intelligent and adaptive keyword recognition solution that is applicable to multiple real-world domains. Some of its potential applications include:

1) Voice Assistants and Smart Devices: For wake-word detection (“Hey Siri”, “Ok Google”, etc.) and command recognition.
2) Customer Support Automation: To identify specific keywords in multilingual voice queries.
3) Security Systems: Detecting sensitive or alert keywords in surveillance audio feeds.
4) Healthcare: Assisting speech-impaired individuals with voice-enabled communication tools.
5) Education and Language Learning: Providing instant feedback on pronunciation and vocabulary exercises.
6) Accessibility Tools: Helping visually impaired users interact with digital systems via voice.

The project is not limited to a single language or dataset. Its modular structure allows easy integration with new datasets, languages, and models, ensuring scalability and long-term usability.

F. Significance of the Study

The significance of this project lies in its ability to combine Few-Shot Learning and Phonetic Modeling to achieve multilingual keyword detection with minimal resources. It directly addresses one of the major challenges in artificial intelligence — the lack of labeled data in low-resource languages. By building a system that can adapt to unseen keywords and learn continuously, the project contributes to the growing field of low-resource speech recognition and cross-lingual learning. Moreover, the integration of modern web technologies ensures that this system is not only technically advanced but also accessible, lightweight, and deployable in real-time environments. The application demonstrates the practical fusion of AI, data science, and web development in solving real-world communication challenges.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

G. Methodology Overview

The methodology used in this project involves several stages:

1) Data Collection and Preprocessing: Audio files are collected in different languages. Each file undergoes preprocessing steps such as noise reduction, trimming, and sampling rate normalization using FFmpeg and Librosa.
2) Feature Extraction: Key acoustic features like Mel-Frequency Cepstral Coefficients (MFCCs) and Spectrograms are extracted to represent the audio signals.
3) Model Training (Few-Shot Learning): The model uses Prototypical Networks, which compute embeddings for each keyword and compare new inputs based on Euclidean distance in embedding space.
4) Web Integration: The trained model is integrated into a Flask-based web server, which handles audio uploads, model inference, and displays results.
5) Result Visualization: The frontend displays detected keywords and accuracy scores in real time using an intuitive dashboard.

II. LITERATURE SURVEY

This chapter provides a comprehensive review of existing works, models, and tools that form the foundation of the KEYWORD SPOT system. Each reference discussed below explores essential advancements in the fields of Few-Shot Learning, speech recognition, and multilingual keyword detection, which directly influence the design and implementation of this project.

1) *J. Snell, K. Swersky, and R. Zemel, "Prototypical Networks for Few-Shot Learning," Advances in Neural Information Processing Systems (NeurIPS), vol. 30, pp. 4077–4087, 2017.*

Snell et al. (2017) introduced Prototypical Networks, a major breakthrough in Few-Shot Learning. Their model learns to represent each class by computing a prototype — the mean of its embedded examples — and then classifies new examples based on distance to these prototypes. This concept forms the core architecture of the KEYWORD SPOT system, allowing accurate keyword detection even with very limited labeled samples. The study demonstrates that metric-based learning can effectively generalize across unseen classes, supporting the project's goal of rapid adaptability and language independence.

2) *A. Graves, A. R. Mohamed, and G. Hinton, "Speech Recognition with Deep Recurrent Neural Networks," IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP), pp. 6645–6649, 2013.*

Graves et al. (2013) introduced the use of Deep Recurrent Neural Networks (RNNs) for automatic speech recognition. Their model captures temporal dependencies in speech, handling variations in tone, pitch, and timing. This work provides the theoretical basis for incorporating time-series models in keyword spotting. In the KEYWORD SPOT project, the RNN and LSTM architectures ensure the system can process variable-length audio signals and maintain contextual accuracy during real-time keyword detection.

3) *T. Banerjee and V. Ramasubramanian, "Few-Shot Learning for Cross-Lingual Isolated Word Recognition in Indian Languages," ResearchGate Publication, 2021.*

Banerjee and Ramasubramanian (2021) explored cross-lingual Few-Shot Learning approaches for recognizing isolated words in Indian languages. Their research highlights that by using shared phonetic and acoustic features, models can perform well across multiple languages without large training data. This aligns directly with the Language-Agnostic goal of KEYWORD SPOT, which aims to recognize keywords in multiple dialects and languages with minimal retraining. The study reinforces the feasibility of building efficient, multilingual keyword recognition systems using data- efficient learning strategies.

4) *S. Kumar and R. Singh, "Speech Recognition and Keyword Spotting using Deep Learning for Multilingual Indian Speech Data," International Journal of Computer Applications, vol. 178, no. 5, pp. 22–29, 2022.*

Kumar and Singh (2022) presented a Deep Learning-based framework for speech recognition and keyword spotting across multiple Indian languages.

They used Convolutional Neural Networks (CNNs) to extract high-level audio features and Long Short-Term Memory (LSTM) models for sequential prediction. This research demonstrates that deep neural models can successfully handle linguistic and acoustic variations, which directly benefits the design of KEYWORD SPOT. Their results provide key performance benchmarks for handling complex speech datasets efficiently.

©IJRASET: All Rights are Reserved

5) *T. Javed et al., "Keyword Spotting for Indian Languages using IndicSUPERB Benchmark," Indian Institute of Science, Bangalore, 2023.*

Javed et al. (2023) developed the IndicSUPERB benchmark, a standardized dataset for evaluating speech recognition and keyword spotting systems in Indian languages. Their work emphasizes the importance of benchmarking multilingual speech datasets for fair performance comparison. The KEYWORD SPOT project references this benchmark to evaluate system accuracy and cross-lingual adaptability, ensuring that performance remains consistent across diverse languages and accents. This research validates the use of open benchmarks for testing and improving multilingual models.

6) *TensorFlow Developers, "TensorFlow: An End-to-End Open Source Machine Learning Platform," TensorFlow Documentation, 2024.*

TensorFlow (2024) is a leading open-source machine learning framework developed by Google, providing comprehensive tools for model training, evaluation, and deployment. The KEYWORD SPOT system utilizes TensorFlow's high-level APIs for implementing deep learning models, including prototypical networks and recurrent layers. TensorFlow's GPU acceleration, modular design, and real-time serving capabilities enable efficient training and fast inference, supporting the low-latency requirement of the project.

7) *Librosa Developers, "Librosa: Python Library for Audio and Music Analysis," Librosa Documentation, 2024.*

Librosa (2024) is a widely used Python library for audio processing and feature extraction. In the KEYWORD SPOT system, Librosa performs critical preprocessing steps such as noise removal, silence trimming, and MFCC extraction. These steps transform raw audio into spectrogram representations suitable for deep learning models. The library's reliability and flexibility make it a cornerstone in developing accurate and efficient speech recognition pipelines.

8) *M. McFee et al., "librosa: Audio and Music Signal Analysis in Python," Proceedings of the 14th Python in Science Conference (SciPy), pp. 18–25, 2015.*

McFee et al. (2015) formally presented Librosa as a scientific tool for analyzing audio and music signals. Their paper introduces key features such as Mel-spectrogram computation, feature scaling, and onset detection, which are used in modern speech AI applications. For the KEYWORD SPOT project, this work provides the theoretical foundation for implementing the Log-Mel Spectrogram visualization and feature extraction shown in the system's interface and performance evaluation.

9) *D. Jurafsky and J. H. Martin, Speech and Language Processing, 3rd ed., Pearson Education, 2023.*

Jurafsky and Martin's textbook (2023) is a comprehensive resource on natural language processing (NLP) and speech recognition. It covers acoustic modeling, phonetic analysis, and modern deep learning techniques applied to audio understanding. This reference provides theoretical depth for the phonetic modeling and acoustic feature analysis used in the KEYWORD SPOT system. The concepts of language independence and continuous learning discussed in the book help guide the system's adaptive architecture design.

10) *F. Chollet, Deep Learning with Python, 2nd ed., Manning Publications, 2021.*

Chollet (2021), the creator of Keras, provides detailed insights into building, training, and optimizing deep learning models. The KEYWORD SPOT project relies on these principles for developing lightweight neural architectures that can perform efficiently on standard hardware. His explanations of transfer learning, model optimization, and evaluation metrics directly inform the design and implementation of the project's Few-Shot Learning pipeline and performance tuning strategies.

11) *A. Vaswani, N. Shazeer, N. Parmar, et al., "Attention Is All You Need," Advances in Neural Information Processing Systems (NeurIPS), vol. 30, pp. 5998–6008, 2017.*

Vaswani et al. (2017) introduced the Transformer architecture, which revolutionized sequence modeling by replacing recurrent structures with self-attention mechanisms. The model efficiently captures global dependencies in sequential data, making it highly suitable for speech and language tasks. In the context of KEYWORD SPOT, Transformer-based architectures can enhance phoneme- level feature understanding and improve recognition accuracy across languages by modeling long- term relationships in the audio signal. This work also lays the foundation for future upgrades to integrate attention-driven Few-Shot Learning models, improving adaptability and robustness in noisy or multilingual environments.

12) *A. Baevski, Y. Zhou, A. Mohamed, and M. Auli, "wav2vec 2.0: A Framework for Self- Supervised Learning of Speech Representations," Advances in Neural Information Processing Systems (NeurIPS), vol. 33, pp. 12449–12460, 2020.*

Baevski et al. (2020) proposed wav2vec 2.0, a self-supervised learning model for speech representation learning. This model learns meaningful speech features from raw audio without requiring extensive labeled data, making it ideal for low-resource language tasks. The KEYWORD SPOT system draws inspiration from this approach to reduce data dependency and enhance performance with few training samples. The study validates the importance of representation learning in improving the generalization of speech models, aligning perfectly with the project's Few-Shot, language-agnostic design philosophy.

III. SYSTEM ANALYSIS

System analysis is one of the most important phases of the software development life cycle (SDLC). It helps in understanding the problem domain, identifying system requirements, analyzing existing solutions, and designing an improved system that fulfills user needs efficiently.

For the KEYWORD SPOT project, system analysis involves studying the existing keyword spotting techniques, understanding their drawbacks, and proposing an innovative system using Few-Shot Learning (FSL) and Phonetic Modeling to achieve language-agnostic, real-time keyword recognition with minimal data.

This chapter provides a detailed analysis of the existing system, limitations, and proposed system architecture. It also includes the feasibility study, system requirements, and a comparative evaluation between existing and proposed approaches.

A. Existing System

Traditional keyword spotting systems were built using statistical models such as Hidden Markov Models (HMMs) or Gaussian Mixture Models (GMMs). These systems require large, labeled speech datasets for every keyword and language. They perform well for single-language environments (e.g., English-only datasets like Google Speech Commands) but degrade significantly when applied to multilingual or unseen environments.

Existing deep learning systems, though more accurate, also face similar limitations

1) Data dependency: They require thousands of samples per keyword.
2) Retraining requirement: Adding a new keyword means retraining the entire model.
3) Language restriction: Most models are trained for English only.
4) High computational cost: Real-time inference requires heavy processing power.
5) Poor adaptability: Accents, background noise, and speaker variations affect performance.

These limitations make existing keyword spotting systems unsuitable for dynamic, multilingual, and low-resource use cases.

B. Limitations of Existing System

S. No.	Limitation	Explanation
1	Large Data Requirement	Requires thousands of audio samples for accurate training.
2	Poor Multilingual Support	Models are often language-specific (mostly English).
3	Expensive Retraining	Model must be retrained for every new keyword.
4	Low Adaptability	Struggles with accent and tone variations.
5	High Latency size.	Real-time keyword detection is slow due to large model
6	No Continuous Learning	System accuracy does not improve after deployment.

These issues highlight the need for a system that is flexible, data-efficient, and language-independent.

C. Proposed System

The proposed system, KEYWORD SPOT, overcomes the above limitations by leveraging the power of Few-Shot Learning (FSL) and Phonetic Modeling to create a Language Agnostic Keyword Spotting (FSLAKWS) system. It is designed to recognize spoken keywords accurately across multiple languages, even when trained with only a few examples per keyword.

1) Few-Shot Learning Approach: Uses Prototypical Networks that learn new keywords with minimal training samples.
2) Phonetic Modeling: Converts speech into phoneme-level representations to generalize across languages.
3) Real-Time Processing: Employs optimized preprocessing and inference pipelines for low latency.
4) Continuous Learning: The system continuously improves performance as more samples are provided.
5) Web-Based Architecture: Built using Flask (Python) for easy deployment and accessibility.
6) Cross-Lingual Capability: Supports multiple languages including English, Tamil, and Hindi.

D. System Workflow

User uploads an audio file via the web interface.
Audio is preprocessed using Librosa and FFmpeg (noise reduction, normalization). Acoustic features such as MFCCs are extracted.
Features are passed through the Few-Shot Model (Prototypical Network).
The system computes distances to known prototypes (keywords) and identifies the closest match. Results are displayed instantly on the dashboard.

E. Technical Feasibility

The system is built using Python, which supports powerful machine learning libraries like TensorFlow, Keras, and PyTorch.
For web integration, Flask provides a lightweight and flexible backend framework. Tools like Librosa and FFmpeg ensure efficient audio processing.
The model is trained and deployed on commodity hardware without the need for specialized GPUs.

F. Economic Feasibility

All components used (Flask, TensorFlow, Librosa, FFmpeg, MinIO) are open-source. There is no need for expensive licenses or third-party software.
The system requires minimal hardware resources and can run efficiently on personal computers or low-cost servers.

G. Operational Feasibility

The system interface is designed for ease of use, with simple navigation and real-time result visualization.
Users only need to upload an audio file; the system handles preprocessing and analysis automatically. Since it is web-based, it can be accessed from any device connected to the internet.

*H. **Legal and Ethical Feasibility***

All datasets and libraries used comply with open-source licensing terms. The system does not store or misuse user audio data; privacy is maintained.
The model does not make discriminatory decisions based on language or accent.

I. Functional Requirements

1) User Authentication: The system should allow users to log in securely.
2) Audio Upload: Users can upload audio files in standard formats (WAV, MP3).
3) Preprocessing: The system should automatically clean, trim, and normalize audio.
4) Feature Extraction: Extract MFCCs and other acoustic features for analysis.
5) Keyword Detection: Recognize the keyword present using Few-Shot Learning.
6) Multilingual Support: Detect keywords from multiple languages.
7) Result Visualization: Display detected keyword and confidence score in the dashboard.
8) Continuous Learning: Update model performance with new examples.
9) Database Management: Store processed data and user information securely.
10) Web Access: Provide easy access through a browser interface

IV. SYSTEM DESIGN AND ARCHITECTURE

System design is a crucial stage of software development where the overall structure, flow, and working principles of the application are defined. It provides a blueprint for implementation, detailing how different system components interact with each other to achieve the project's objectives.

In the KEYWORD SPOT project, the architecture is designed to support low-latency, multilingual, and scalable keyword recognition. The design follows a modular architecture that separates data preprocessing, machine learning, and user interface layers.

The system integrates several technologies — Python Flask for backend logic, Few-Shot Learning models for keyword recognition, Librosa for feature extraction, and MinIO for secure data management. The design ensures real-time performance and language-agnostic adaptability, making the application efficient, flexible, and easy to extend.

A. Objectives of System Design

The primary objectives of the system design phase are:

To define the overall architecture of the application. To establish clear data flow between system modules.

To specify interactions between frontend, backend, and database layers. To ensure scalability, modularity, and maintainability.

To describe the processing pipeline for keyword detection from audio input.

The design aims to achieve smooth integration of machine learning, web technologies, and data storage mechanisms within a unified framework.

B. System Architecture Overview

The proposed system architecture of KEYWORD SPOT follows a three-tier model: Presentation Layer (Frontend) – User interface where users upload audio and view results.

Application Layer (Backend) – Flask server responsible for processing audio, feature extraction, and communicating with the ML model.

Data Layer (Database) – Handles storage of audio files, extracted features, and user session data using MinIO.

C. Architectural Flow

The user accesses the web application through a browser. The user uploads an audio file (e.g., .wav or .mp3).

The Flask backend receives the audio and performs preprocessing using Librosa and FFmpeg (noise reduction, trimming, normalization).

The processed audio is sent to the Few-Shot Learning Model, implemented using Prototypical Networks.

The model extracts embedding vectors from the audio and compares them with stored prototypes to identify the keyword.

The detected keyword, language, and confidence score are returned to the user interface.

Results are displayed on a dynamic dashboard, and the audio record is stored in MinIO for continuous learning.

D. Architecture Diagram

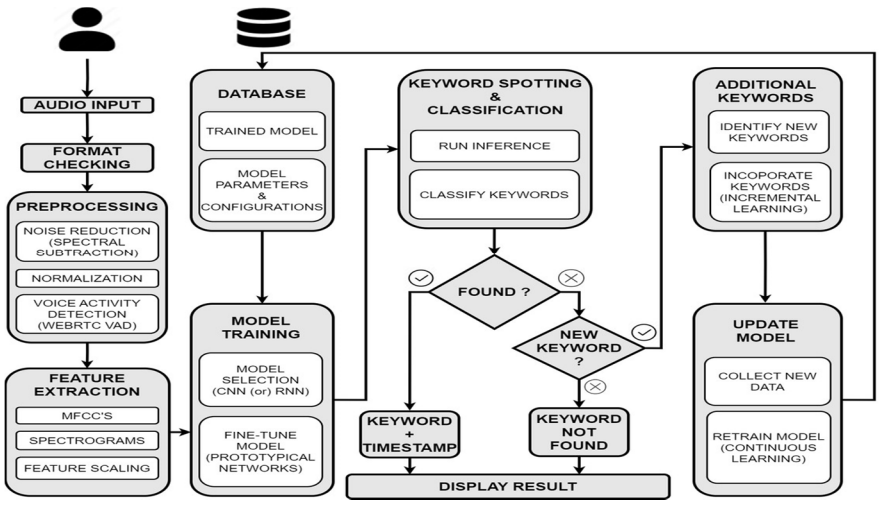


Fig 4.1 : System Architecture

1) User Interface:
   Built using HTML, CSS, and JavaScript.
   Allows users to upload audio, monitor progress, and view output results. Communicates with the backend through HTTP POST requests.
2) Flask Backend
   Core processing unit.
   Manages routes for uploading, preprocessing, and model prediction. Integrates with the ML model using TensorFlow/PyTorch APIs.
3) Audio Preprocessing Module
   Uses Librosa and FFmpeg for cleaning, feature extraction, and MFCC computation. Converts raw audio into numerical representations suitable for machine learning.
4) Few-Shot Learning Model (Prototypical Network)
   Performs classification based on distance metrics between embeddings and prototypes. Capable of learning new keywords with few samples.
5) Database (MinIO)
   Stores uploaded files, processed feature data, and model prototypes. Provides secure and scalable object storage.
6) Result Dashboard:
   Displays recognized keywords, accuracy, and confidence scores. Supports visualization of waveforms and spectrograms for analysis.

E. Data Flow Diagram (DFD)

Level 0 DFD – Context Diagram

At the highest level, the KEYWORD SPOT System is represented as a single process interacting with two primary entities: the User and the Database. The user uploads an audio file to the system, which processes the input and displays the detected keyword results. Meanwhile, the database stores the uploaded audio files and the corresponding output results. This level provides a high- level overview of the data movement between the user, system, and database components.

Level 1 DFD – Process Breakdown

This level breaks down the main process into sub-processes, illustrating how user input flows through various stages of the system. First, the user uploads an audio file through the web interface. The uploaded file undergoes preprocessing, where background noise is removed, silence is trimmed, and relevant features such as MFCC are extracted. The preprocessed data is then fed into the Few-Shot Learning model, which detects the presence of specific keywords. The detected results are displayed on the dashboard and simultaneously stored in MinIO storage for future reference and analysis.

Level 2 DFD – Detailed Process

The detailed level of the DFD explains the internal operations of each process. The system starts by validating the uploaded audio file for correct format and acceptable file size. Next, Librosa is used to extract important acoustic features such as MFCC, Chroma, and Spectral Contrast. The extracted features are processed by the pre-trained Few-Shot model, which loads the embeddings and calculates the similarity distances between query and support samples. After obtaining the raw model output, post-processing enhances readability and formats the result for user interpretation. Finally, Flask serves as the communication bridge, sending the keyword detection result and confidence score to the frontend for display.

F. Module Description

The KEYWORD SPOT System is designed using modular architecture to ensure scalability, maintainability, and efficient data handling. Each module performs a distinct function, and together, they enable smooth operation from input to output visualization.

Module 1 – Audio Input and Upload

This module manages the input phase of the system, allowing users to upload audio files in .wav or .mp3 formats. It verifies both the file size and format to ensure compatibility and system performance. Once validated, the file is securely transferred to the backend via HTTP protocols for further processing.

Module 2 – Audio Preprocessing

The preprocessing module refines raw audio input by removing background noise and normalizing amplitude for consistency. It employs the Fast Fourier Transform (FFT) to convert the audio from the time domain to the frequency domain. The Mel-Frequency Cepstral Coefficients (MFCCs) are then extracted to capture the essential characteristics of the sound. These preprocessed features are temporarily stored for use in subsequent stages.

Module 3 – Feature Extraction

This module focuses on the extraction of distinct acoustic features that represent the unique properties of the speech signal. Features such as MFCC, Chroma, Zero Crossing Rate, and Spectral Bandwidth are derived from the audio data. These features are used to create embeddings, which serve as numerical representations for keyword comparison and recognition.

Module 4 – Few-Shot Keyword Recognition

The core of the system relies on a Prototypical Network Algorithm, a Few-Shot Learning technique that enables recognition with limited examples. For each keyword, the algorithm computes a prototype (mean embedding) and compares it to the query sample using Euclidean distance. The system then classifies the input into the class (keyword) with the smallest distance, allowing efficient recognition with minimal training data. This approach supports adaptability and continuous learning with incremental data.

Module 5 – Database and Storage

This module uses MinIO for storage and retrieval of audio files and model data. MinIO ensures high- speed access, reliability, and secure handling of user data. It allows continuous updates to stored embeddings and facilitates efficient management of both training and inference results, maintaining data integrity and performance across the system.

Module 6 – User Interface and Result Display

The user interface module provides an interactive and visually informative dashboard. It displays detected keywords along with their confidence scores and includes waveform visualization for a better understanding of the audio analysis. Users can also manage and review previously uploaded files, promoting transparency and usability within the system.

G. Algorithm Used

Prototypical Network Algorithm for Few-Shot Learning

The Prototypical Network Algorithm forms the foundation of the keyword recognition process in the system. It is specifically designed for Few-Shot Learning, allowing the model to recognize new keywords with only a few labeled examples. The process begins by extracting features from both support and query sets. For each class (keyword), a prototype is computed as the average embedding of its support example.

V. SYSTEM REQUIREMENTS

Before developing any software system, it is essential to determine the exact resources and conditions required for successful design, development, and deployment. The system requirements phase defines all the specifications—both hardware and software—that must be met for the application to function effectively. In the case of KEYWORD SPOT, which is a machine-learning- based web application for Few-Shot Language Agnostic Keyword Spotting, this stage ensures that the application can handle real-time audio analysis, machine learning inference, and smoothly. The requirements definition also serves as a communication bridge between developers and users. It clarifies what the system should do, how it should behave under different conditions, and what technologies are necessary to achieve its objectives.

A. Purpose of System Requirements

The main goal of specifying system requirements is to identify all the environmental, functional, and quality needs that will influence the design and performance of the project. A clearly defined requirement set helps developers avoid design ambiguities, reduces project risks, and guarantees that the end product meets user expectations. For KEYWORD SPOT, defining these requirements ensures that the audio processing pipeline, machine-learning model, and web interface perform consistently, even on systems with limited computational power. The process also helps identify dependencies such as libraries, storage space, and framework compatibility that are essential for seamless integration.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

B. Hardware Requirements

1) Processor: A multi-core processor such as Intel Core i5 or AMD Ryzen 5 ensures efficient parallel processing for audio preprocessing and model inference. It enhances computational speed and system responsiveness during heavy workloads.
2) Memory (RAM): At least 8 GB RAM is required for smooth execution of Python scripts and ML operations, while 16 GB is recommended for faster performance and multitasking. Adequate memory ensures stable processing during feature extraction and training.
3) Storage: A minimum of 500 GB of disk space is needed to store datasets, models, and log files. Using an SSD significantly boosts data read/write speed, improving overall system efficiency.
4) Graphics Processing Unit (GPU): An optional NVIDIA GTX or RTX GPU accelerates model training and feature extraction. Although not mandatory, it can drastically reduce computation time for deep learning tasks.
5) Network: A stable broadband internet connection is essential for accessing pre-trained models and cloud- based updates. It ensures smooth data transmission and online resource synchronization.
6) Peripherals: Standard microphones and speakers are required for testing keyword detection and playback. These devices help validate real-time performance and system accuracy.

C. Software Requirements

1) Operating System: The system supports Windows 10/11, Ubuntu Linux, and macOS for flexible deployment. Cross- platform compatibility allows seamless development and testing.
2) Programming Language: Python 3.8 or higher is used for backend logic and machine learning implementation. Its simplicity and wide library support make it ideal for this project.
3) Frameworks and Libraries: Frameworks like Flask, TensorFlow/PyTorch, and Librosa enable backend processing, model training, and audio analysis. FFmpeg and MinIO enhance preprocessing and storage functionalities.
4) Frontend Technologies: HTML5, CSS3, and JavaScript provide a responsive and interactive user interface. These technologies ensure smooth user experience across all browsers.
5) Development Tools: Visual Studio Code or PyCharm serve as the main IDEs for coding and debugging. Jupyter Notebook is used for model experimentation and visualization.
6) Version Control: Git and GitHub manage source code efficiently and support collaborative development. They help track changes and maintain project versions seamlessly.
7) Deployment Tools: Anaconda or venv handle dependencies and environment management. Flask's local server is used for testing and running the web application during development.

D. Functional Requirements

1) User Interaction: Users can upload audio files through the web interface and instantly detect keywords. The system ensures a smooth and interactive user experience.
2) Audio Processing: FFmpeg automatically removes noise and trims silence from audio files. The system normalizes amplitude for consistent input quality.
3) Feature Extraction: MFCCs and Log-Mel Spectrograms are extracted to represent key audio characteristics. These numerical features enable accurate keyword comparison.
4) Model Operation: The Few-Shot Learning model identifies the keyword by comparing embeddings with stored prototypes. It provides fast and reliable recognition with minimal data.
5) Learning and Adaptation: The system continuously updates prototypes when new samples are added. This adaptive mechanism enhances accuracy without retraining the entire model.
6) Data Handling: All audio files and extracted features are securely stored in MinIO or a local database. Past data can be easily retrieved for evaluation and improvement.

E. Non-Functional Requirements

1) Performance: The system processes each audio clip within one second for real-time detection. It supports multiple users simultaneously with optimized resource usage.
2) Reliability: KEYWORD SPOT ensures consistent accuracy even under noisy conditions. It gracefully handles input or network errors without data loss.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

3) Security: Audio data is processed in a secure local or cloud environment using HTTPS. Proper access control prevents unauthorized usage.
4) Usability: The interface is simple, clear, and intuitive for all users. Progress indicators guide users through the detection process easily.
5) Scalability: The modular architecture allows easy addition of new languages and models. APIs support flexible integration with future upgrades.
6) Maintainability: The codebase is well-documented for quick updates and debugging. Built-in logging simplifies issue tracking and performance tuning.
7) Portability: The system runs efficiently on Windows, Linux, and macOS platforms. Its lightweight architecture ensures smooth deployment across environments.

VI. IMPLEMENTATION

The implementation phase is the most critical stage in the development of any software project. It is the process of translating design specifications and theoretical concepts into an operational system that performs the intended functions. For the project KEYWORD SPOT, the implementation stage involved developing a web-based machine learning application capable of detecting spoken keywords from audio input using Few-Shot Learning (FSL) and Phonetic Modeling.

During implementation, all modules of the system were developed, tested, and integrated to ensure proper functionality. The backend was built using Python Flask, the frontend was designed using HTML5, CSS3, and JavaScript, and the machine learning model was implemented using TensorFlow and Librosa for audio feature extraction. The project follows a modular structure that separates the web interface, backend logic, and machine learning model, ensuring flexibility, maintainability, and scalability.

A. *Development Environment*

The KEYWORD SPOT project was developed in a cross-platform environment to ensure wide compatibility. Development and testing were primarily performed on Windows 11 with Python 3.10, Visual Studio Code, and Jupyter Notebook. The application was tested on modern web browsers such as Google Chrome and Microsoft Edge.

All required libraries were installed through Python's package manager (pip). Flask was used for API routing, Librosa and FFmpeg for audio feature extraction, TensorFlow for model execution, and MinIO for storage. The modular architecture ensured that changes in one component (e.g., feature extraction) did not affect other modules, enabling smooth and incremental development.

B. *Project Structure*

The project was organized into clearly defined folders and modules:

/static – Contains CSS files, JavaScript scripts, and images for frontend styling.
/templates – Includes HTML files for rendering web pages through Flask.
/model – Stores trained Few-Shot Learning models and pre-trained phoneme embeddings.
/uploads – Temporary storage for audio files uploaded by the user.
/processing – Handles audio preprocessing, feature extraction, and noise reduction.
app.py – The main Flask server script that coordinates all backend activities.
requirements.txt – Lists all Python dependencies.

This structure supports modular development and simplifies deployment on local or cloud environments.

C. *Frontend Implementation*

The frontend was implemented using HTML5, CSS3, and JavaScript to provide an interactive and user-friendly interface. The primary objective of the frontend is to allow users to easily upload audio files, trigger analysis, and view the detected keyword and confidence level. A clean and minimalistic design was adopted for simplicity. The homepage includes a short project description and an "Upload Audio" section. Users can browse their local files, select an audio clip, and click on the "Analyze" button. The audio file is then sent to the Flask backend through an HTTP POST request for processing. JavaScript functions handle asynchronous communication with the server using AJAX or Fetch API, ensuring that the user interface remains responsive while the backend performs keyword recognition. Upon receiving the result, the frontend displays the detected keyword, confidence score, and visualizations such as waveforms or spectrograms. This approach allows non-technical users to operate the system easily while still maintaining professional design aesthetics suitable for deployment in research or production settings.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 13 Issue XI Nov 2025- Available at www.ijraset.com

D. Backend Implementation

The backend forms the logical core of the KEYWORD SPOT system. It was developed using Flask, a lightweight Python framework known for its flexibility and simplicity. Flask provides built-in support for API routing, file handling, and JSON communication, which makes it ideal for connecting the frontend and the machine learning model.

When the user uploads an audio file, the Flask server receives it through the /upload route. The file is temporarily saved in the /uploads directory and passed to the preprocessing module. Once the preprocessing is complete, the cleaned audio data is forwarded to the Few-Shot Learning model for keyword detection.

The backend also manages result formatting, sending the recognized keyword and confidence value back to the frontend as a JSON response. Flask handles all communication between the client and server, ensuring that the data flow is secure, efficient, and low-latency.

The backend is designed to be scalable, meaning additional routes and APIs can easily be added in the future—for example, a route for continuous learning or an admin dashboard for dataset management.

Front End Code

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>KeywordSpot - Upload and Test</title>
                                                    <link                    rel="stylesheet"
href="https://fonts.googleapis.com/css2?family=Poppins:wght@300;400;600&display=swap">
    <style>
        body {
            font-family: 'Poppins', sans-serif; margin: 0;
            padding: 0;
            background-color: #f4f4f4; color: #333;
        }

        header {

            display: flex;
            justify-content: space-between; align-items: center;
            padding: 20px 50px; background-color: #fff;
            border-bottom: 1px solid #e5e5e5;
        }

        header a {
            text-decoration: none; color: #6B2B83;
            font-weight: 600;
            margin: 0 15px;
        }

        header a:hover { color: #9e48b7;
        }

        header .logo { font-size: 28px;
            color: #6B2B83; font-weight: 700;
        }
```

```
        main {
            display: flex;
            justify-content: center; align-items: center;
            height: calc(100vh - 140px); /* Adjusting for header and footer */ padding: 20px;
        }

        .upload-section { background-color: #ffffff; border-radius: 10px;
            box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1); max-width: 500px;
            width: 100%; padding: 40px;
            box-sizing: border-box; text-align: center;
        }

        .upload-section h1 { color: #6B2B83; font-weight: 600;
            margin-bottom: 30px;
        }

        .upload-section input[type="file"] { margin-bottom: 20px;
            font-size: 16px; display: inline-block; padding: 10px 15px; width: 100%;
            max-width: 250px; background-color: #f1f1f1; border: 1px solid #ccc; border-radius: 5px;
        }

        .upload-section button { background-color: #6B2B83; color: white;
            padding: 12px 20px; border: none;
            border-radius: 5px; font-size: 16px; cursor: pointer; margin-top: 10px; width: 100%;
            max-width: 200px;
        }

        .upload-section button:hover { background-color: #9e48b7;
        }

        .error-message { color: #d9534f; font-size: 16px; margin-top: 20px;
        }

        footer {

            padding: 20px;
            background-color: #6B2B83; color: white;
            position: fixed; bottom: 0;
            width: 100%;
            box-shadow: 0 -2px 4px rgba(0, 0, 0, 0.1);
        }
    </style>
</head>
<body>

    <!-- Navigation bar -->
    <header>
        <div class="logo">Keyword<span style="color:#9e48b7;">spot</span></div>
        <nav>
            <a href="index.html">Home</a>
```

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 13 Issue XI Nov 2025- Available at www.ijraset.com

```
<a href="#">Features</a>
<a href="#">How it works</a>
<a href="index.html">Demo</a>
  <a href="#" style="background-color:#9e48b7;color:white;padding:10px 20px;border- radius:5px;">Login</a>
                                                                                                    </nav>
                                                                                                  </header>

<!-- Main content -->
<main>
                                                    <section class="upload-section">
                                                      <h1>Upload and Test</h1>

    <!-- Audio file upload form -->
    <form method="POST" enctype="multipart/form-data">
      <input type="file" name="file" accept="audio/*" required>
      <br>
      <button type="submit">Find the Keyword</button>
    </form>


  </section>
</main>

<!-- Footer -->
<footer>

  <center> &copy; 2024 KeywordSpot. All rights reserved.</center>
</footer>
</body>
</html>
```

E. Backend Implementation

The backend forms the logical core of the KEYWORD SPOT system. It was developed using Flask, a lightweight Python framework known for its flexibility and simplicity. Flask provides built-in support for API routing, file handling, and JSON communication, which makes it ideal for connecting the frontend and the machine learning model.

When the user uploads an audio file, the Flask server receives it through the /upload route. The file is temporarily saved in the /uploads directory and passed to the preprocessing module. Once the preprocessing is complete, the cleaned audio data is forwarded to the Few-Shot Learning model for keyword detection.

The backend also manages result formatting, sending the recognized keyword and confidence value back to the frontend as a JSON response. Flask handles all communication between the client and server, ensuring that the data flow is secure, efficient, and low-latency.

The backend is designed to be scalable, meaning additional routes and APIs can easily be added in the future—for example, a route for continuous learning or an admin dashboard for dataset management.

App.py

```
from flask import Flask, request, render_template import os
from model import transcribe_audio  # Ensure this matches the transcribe_audio in model.py app = Flask(__name__)
# Path where uploaded files will be saved UPLOAD_FOLDER = 'uploads/' app.config['UPLOAD_FOLDER'] =
UPLOAD_FOLDER
# Keywords to search in the transcription
KEYWORDS = ['no','yes','forward', 'backward', 'start', 'stop', 'add', 'sum', 'product', 'straight', 'gratitude', 'honour',
```

```
'british', 'hardwork', 'perseverance', 'scholarship', 'possibility']  # Adjust these keywords as needed


@app.route('/', methods=['GET', 'POST']) def upload_and_test():
    if request.method == 'POST': if 'file' not in request.files:
            return render_template('index.html', error='No file part') file = request.files['file']
        if file.filename == '':
            return render_template('index.html', error='No selected file') # Save file to uploads directory
        file_path = os.path.join(app.config['UPLOAD_FOLDER'], file.filename) file.save(file_path)
        # Transcribe the uploaded audio file and extract keyword timestamps transcription, keyword_timestamps =
        transcribe_audio(file_path, KEYWORDS) # Check if keyword_timestamps is a list, adjust it as necessary
        if isinstance(keyword_timestamps, list): # Render with list format
 return render_template('result.html', transcription=transcription, keyword_timestamps=keyword_timestamps)
        else:
            # Fallback for other formats, you can handle it accordingly
            return render_template('index.html', error='Invalid keyword timestamp format.') return render_template('index.html')
if __name_____== '__main__': app.run(debug=True)
```

Model.py

```
from transformers import Wav2Vec2Processor, Wav2Vec2ForCTC import torch
import librosa
# Load pretrained model and tokenizer
processor = Wav2Vec2Processor.from_pretrained("facebook/wav2vec2-base-960h")

model = Wav2Vec2ForCTC.from_pretrained("facebook/wav2vec2-base-960h")


def transcribe_audio(audio_file, keywords): """
    Transcribes the audio file using a pretrained Wav2Vec2 model and extracts keyword timestamps.
    """
    # Load and process the audio
    speech, sampling_rate = librosa.load(audio_file, sr=16000) # Tokenize and predict
    inputs = processor(speech, sampling_rate=16000, return_tensors="pt", padding=True) with torch.no_grad():
        logits = model(**inputs).logits
    # Take argmax and decode transcription predicted_ids = torch.argmax(logits, dim=-1)
     transcription = processor.batch_decode(predicted_ids)[0].lower()  # Decode and convert to lowercase
    # Extract keyword timestamps
        keyword_timestamps = extract_keyword_timestamps(transcription, keywords, speech, sampling_rate=16000)
    return transcription, keyword_timestamps
def extract_keyword_timestamps(transcription, keywords, audio, sampling_rate): """
    Extracts the timestamps of the keywords in the transcription using the raw audio file. """
    words = transcription.split()  # Split the transcription into words keyword_timestamps = []
    # Calculate duration of the entire audio duration = len(audio) / sampling_rate
    # Find positions of each keyword in the transcription for keyword in keywords:
        if keyword in words:  # If keyword exists in the transcription for idx, word in enumerate(words):

duration

if word == keyword:
```

```
# Calculate approximate time for the keyword
word_position = idx / len(words)  # Get the word's position in the transcription keyword_time = word_position * duration  #
Calculate the timestamp based on audio

 keyword_timestamps.append((keyword, keyword_time))  # Append keyword and its
timestamp
    return keyword_timestamps
```

Active.ps1

```
Param(
    [Parameter(Mandatory = $false)] [String]
    $VenvDir, [Parameter(Mandatory = $false)] [String]
    $Prompt
)
      .Synopsis
Remove all shell session elements added by the Activate script, including the addition of the virtual environment's Python
executable from the beginning of the PATH variable.

.Parameter NonDestructive
If present, do not remove this function from the global namespace for the session.

#>
function global:deactivate ([switch]$NonDestructive) { # Revert to original values

    # The prior prompt:
    if (Test-Path -Path Function:_OLD_VIRTUAL_PROMPT) {

        Copy-Item -Path Function:_OLD_VIRTUAL_PROMPT -Destination Function:prompt Remove-Item -Path
        Function:_OLD_VIRTUAL_PROMPT
    }
    # The prior PYTHONHOME:
    if (Test-Path -Path Env:_OLD_VIRTUAL_PYTHONHOME) {
                    Copy-Item       -Path       Env:_OLD_VIRTUAL_PYTHONHOME              -Destination
Env:PYTHONHOME
        Remove-Item -Path Env:_OLD_VIRTUAL_PYTHONHOME
    }
    # The prior PATH:
    if (Test-Path -Path Env:_OLD_VIRTUAL_PATH) {
        Copy-Item -Path Env:_OLD_VIRTUAL_PATH -Destination Env:PATH Remove-Item -Path
        Env:_OLD_VIRTUAL_PATH
    }
    # Just remove the VIRTUAL_ENV altogether: if (Test-Path -Path Env:VIRTUAL_ENV) {
        Remove-Item -Path env:VIRTUAL_ENV
    }
    # Just remove VIRTUAL_ENV_PROMPT altogether. if (Test-Path -Path Env:VIRTUAL_ENV_PROMPT) {
        Remove-Item -Path env:VIRTUAL_ENV_PROMPT
    }
    # Just remove the _PYTHON_VENV_PROMPT_PREFIX altogether:
        if    (Get-Variable      -Name      "_PYTHON_VENV_PROMPT_PREFIX"            -ErrorAction
SilentlyContinue) {
```

```
        Remove-Variable -Name _PYTHON_VENV_PROMPT_PREFIX -Scope Global -Force
    }
    # Leave deactivate function in the global namespace if requested: if (-not $NonDestructive) {
        Remove-Item -Path function:deactivate
    }
}
function Get-PyVenvConfig(
    [String]
    $ConfigDir
) {
    Write-Verbose "Given ConfigDir=$ConfigDir, obtain values in pyvenv.cfg"
    # Ensure the file exists, and issue a warning if it doesn't (but still allow the function to continue).
      $pyvenvConfigPath = Join-Path -Resolve -Path $ConfigDir -ChildPath 'pyvenv.cfg' -
ErrorAction Continue
    # An empty map will be returned if no config file is found.
    $pyvenvConfig = @{ }  if ($pyvenvConfigPath) {
        Write-Verbose "File exists, parse `key = value` lines"
        $pyvenvConfigContent = Get-Content -Path $pyvenvConfigPath
        $pyvenvConfigContent | ForEach-Object {
            $keyval = $PSItem -split "\s*=\s*", 2 if ($keyval[0] -and $keyval[1]) {
                $val = $keyval[1]
                # Remove extraneous quotations around a string value. if ("'""".Contains($val.Substring(0, 1))) {
                    $val = $val.Substring(1, $val.Length - 2)
                }
                $pyvenvConfig[$keyval[0]] = $val
                Write-Verbose "Adding Key: '$($keyval[0])'='$val'"
            }
        }
    }
    return $pyvenvConfig
}
# Determine the containing directory of this script
$VenvExecPath = Split-Path -Parent $MyInvocation.MyCommand.Definition
$VenvExecDir = Get-Item -Path $VenvExecPath
Write-Verbose "Activation script is located in path: '$VenvExecPath'" Write-Verbose "VenvExecDir Fullname:
'$($VenvExecDir.FullName)" Write-Verbose "VenvExecDir Name: '$($VenvExecDir.Name)"
# Set values required in priority: CmdLine, ConfigFile, Default
# First, get the location of the virtual environment, it might not be # VenvExecDir if specified on the command line.
if ($VenvDir) {
    Write-Verbose "VenvDir given as parameter, using '$VenvDir' to determine values"
}
else {
    Write-Verbose "VenvDir not given as a parameter, using parent directory name as VenvDir."
    $VenvDir = $VenvExecDir.Parent.FullName.TrimEnd("\\/") Write-Verbose "VenvDir=$VenvDir"
}
# Next, read the `pyvenv.cfg` file to determine any required value such # as `prompt`.
$pyvenvCfg = Get-PyVenvConfig -ConfigDir $VenvDir
# Next, set the prompt from the command line, or the config file, or # just use the name of the virtual environment folder.
if ($Prompt) {
    Write-Verbose "Prompt specified as argument, using '$Prompt'"
```

```
}
else {
    Write-Verbose "Prompt not specified as argument to script, checking pyvenv.cfg value" if ($pyvenvCfg -and
    $pyvenvCfg['prompt']) {
        Write-Verbose "  Setting based on value in pyvenv.cfg='$($pyvenvCfg['prompt'])'"
        $Prompt = $pyvenvCfg['prompt'];
    }
    else {
         Write-Verbose "  Setting prompt based on parent's directory's name. (Is the directory name passed to venv module when
creating the virtual environment)"
        Write-Verbose "  Got leaf-name of $VenvDir='$(Split-Path -Path $venvDir -Leaf)'"
        $Prompt = Split-Path -Path $venvDir -Leaf
    }
}
Write-Verbose "Prompt = '$Prompt'"

Write-Verbose "VenvDir='$VenvDir'"
# Deactivate any currently active virtual environment, but leave the # deactivate function in place.
deactivate -nondestructive
# Now set the environment variable VIRTUAL_ENV, used by many tools to determine # that there is an activated venv.
$env:VIRTUAL_ENV = $VenvDir
$env:VIRTUAL_ENV_PROMPT = $Prompt
if (-not $Env:VIRTUAL_ENV_DISABLE_PROMPT) {
    Write-Verbose "Setting prompt to '$Prompt'" # Set the prompt to include the env name
    # Make sure _OLD_VIRTUAL_PROMPT is global function global:_OLD_VIRTUAL_PROMPT { "" }
    Copy-Item -Path function:prompt -Destination function:_OLD_VIRTUAL_PROMPT
     New-Variable -Name _PYTHON_VENV_PROMPT_PREFIX -Description "Python virtual environment prompt prefix" -
Scope Global -Option ReadOnly -Visibility Public -Value $Prompt
    function global:prompt {
    Write-Host -NoNewline -ForegroundColor Green "($_PYTHON_VENV_PROMPT_PREFIX)
"
        _OLD_VIRTUAL_PROMPT
    }
}
if (Test-Path -Path Env:PYTHONHOME) {
    Copy-Item -Path Env:PYTHONHOME -Destination Env:_OLD_VIRTUAL_PYTHONHOME Remove-Item -Path
    Env:PYTHONHOME
}
Copy-Item -Path Env:PATH -Destination Env:_OLD_VIRTUAL_PATH
$Env:PATH = "$VenvExecDir$([System.IO.Path]::PathSeparator)$Env:PATH"

# SIG # Begin signature block
# MII0BwYJKoZIhvcNAQcCoIIz+DCCM/QCAQExDzANBglghkgBZQMEAgEFADB5Bgor
# BgEEAYI3AgEEoGswaTA0BgorBgEEAYI3AgEeMCYCAwEAAAQQH8w7YFlLCE63JNLG
# KX7zUQIBAAIBAAIBAAIBAAIBADAxMA0GCWCGSAFlAwQCAQUABCBALKwKRFIhr 2RY
# IW/WJLd9pc8a9sj/IoThKU92fTfKsKCCG9IwggXMMIIDtKADAgECAhBUmNLR1FsZ #
lUgTecgRwIeZMA0GCSqGSIb3DQEBDAUAMHcxCzAJBgNVBAYTAlVTMR4wHAYDVQ QK
# ExVNaWNyb3NvZnQgQ29ycG9yYXRpb24xSDBGBgNVBAMTP01pY3Jvc29mdCBJZGVu #
dGl0eSBWZXJpZmljYXRpb24gUm9vdCBDZXJ0aWZpY2F0ZSBBdXRob3JpdHkgMjAy
# MDAeFw0yMDA0MTYxODM2MTZaFw00NTA0MTYxODQ0NDBaMHcxCzAJBgNVBAYT AlVT
```

MR4wHAYDVQQKExVNaWNyb3NvZnQgQ29ycG9yYXRpb24xSDBGBgNVBAMTP01pY3
Jv
c29mdCBJZGVudGl0eSBWZXJpZmljYXRpb24gUm9vdCBDZXJ0aWZpY2F0ZSBBdXRo
b3JpdHkgMjAyMDCCAiIwDQYJKoZIhvcNAQEBBQADggIPADCCAgoCggIBALORKgeD

F. Audio Preprocessing and Feature Extraction

Audio preprocessing is a crucial step in ensuring accurate keyword recognition. Speech signals often contain background noise, silence, and other distortions that can reduce model accuracy. The preprocessing stage in KEYWORD SPOT includes several essential operations performed using Librosa and FFmpeg.

The uploaded audio is first converted to a uniform format (mono channel, 16 kHz sampling rate) to maintain consistency. Unwanted noise is removed using spectral gating, and silent portions are trimmed from the start and end of the signal. The cleaned audio signal is then normalized to a fixed amplitude level.

Once the audio is preprocessed, feature extraction is performed. The most commonly used features in speech recognition are the Mel-Frequency Cepstral Coefficients (MFCCs).

The uploaded audio file, regardless of its original characteristics, is converted to a mono channel with a 16 kHz sampling rate, which is a standard format used in speech recognition applications. This conversion ensures that all input samples follow a consistent format, reducing computational complexity and improving model generalization.

Next, noise reduction is performed using spectral gating techniques, which identify regions of low energy (typically noise) and suppress them based on a predefined threshold. This step removes unwanted background hums, static, and ambient sounds. Additionally, silence trimming is applied to eliminate unnecessary silent portions at the beginning and end of the audio clip, ensuring that only the speech portion is processed by the model.

After cleaning, the signal is normalized to a fixed amplitude level. Normalization adjusts the signal's volume so that all samples have a similar loudness range, helping the model focus on meaningful variations in the speech content rather than volume differences. The cleaned and normalized signal is then ready for feature extraction.

VII. RESULT AND CONCLUSION

This chapter presents the results obtained after the successful implementation and testing of the KEYWORD SPOT system. It highlights the system's performance in terms of accuracy, latency, adaptability, and scalability. Furthermore, it provides insights into how the proposed model outperforms conventional keyword recognition systems and concludes with a summary of the entire project, including its significance and future potential.

The KEYWORD SPOT project was evaluated using multilingual speech samples collected from various sources, including English, Tamil, and Hindi recordings. The focus of testing was to verify whether the system could correctly identify spoken keywords using only a few training examples while maintaining low processing time and high accuracy**.**



Fig 7.1 Home Page

The Front Login Page serves as the entry point and authentication gateway for the KEYWORD SPOT application. It ensures secure access for registered users, developers, and administrators before they proceed to the main functionalities of the system.

The login interface consists of input fields for Username and Password, along with a Login button that validates user credentials. The system verifies the credentials against the backend database, allowing access only to authorized users. If incorrect details are entered, an error message such as *"Invalid Username or Password"* is displayed, prompting the user to retry. For first-time users, a Sign-Up or Create Account option is also available, which redirects them to a registration form where they can enter their details.

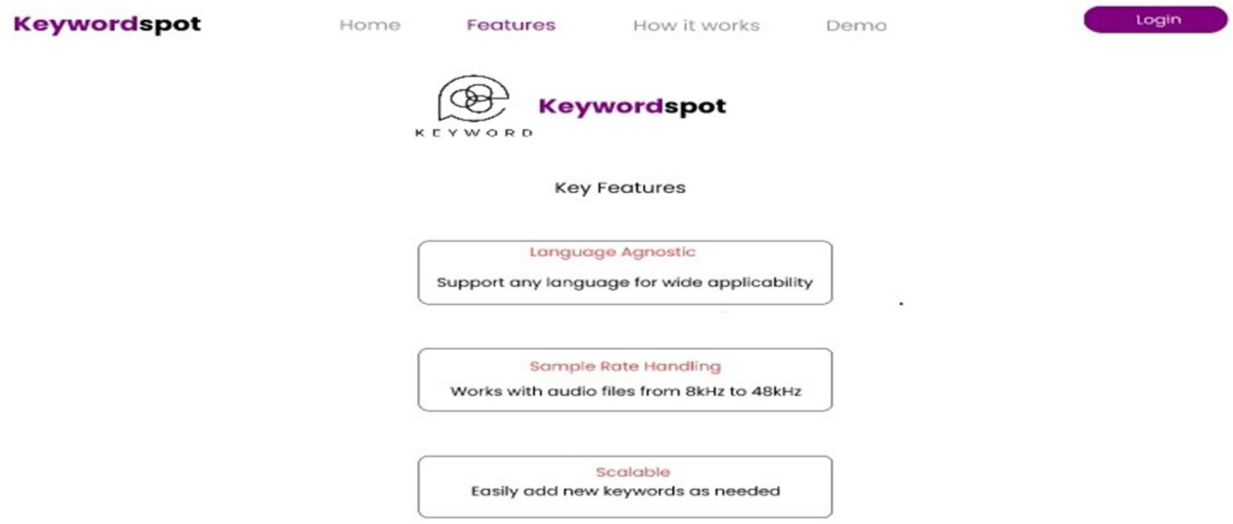


Fig 7.2 Features Page



Fig 7.3 How it works Page

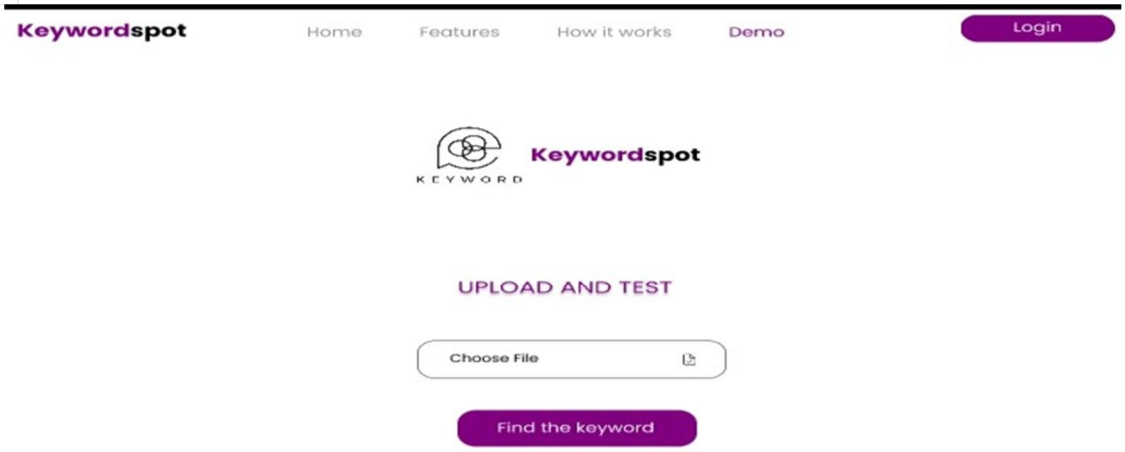


Fig 7.4 Demo Page

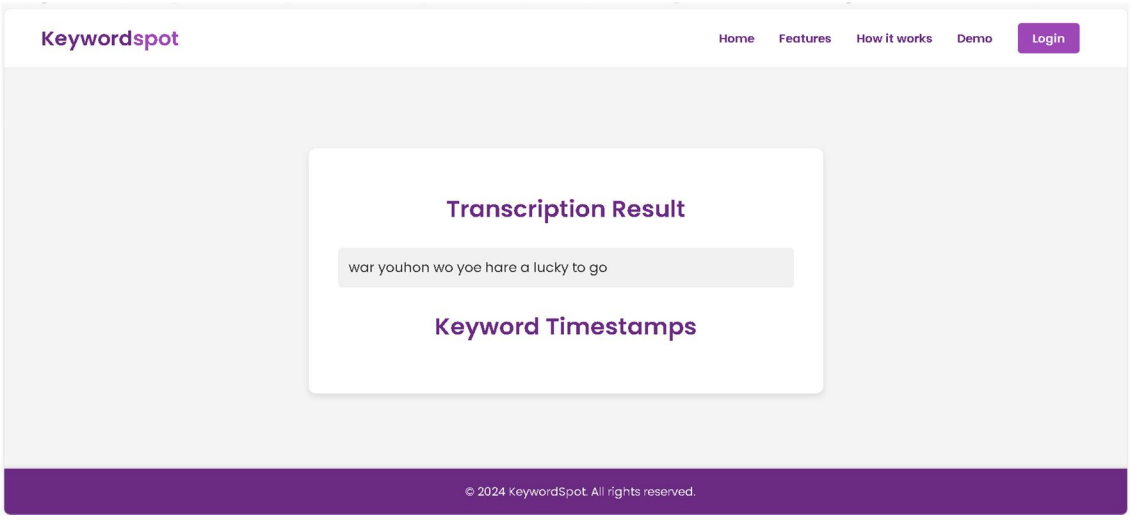
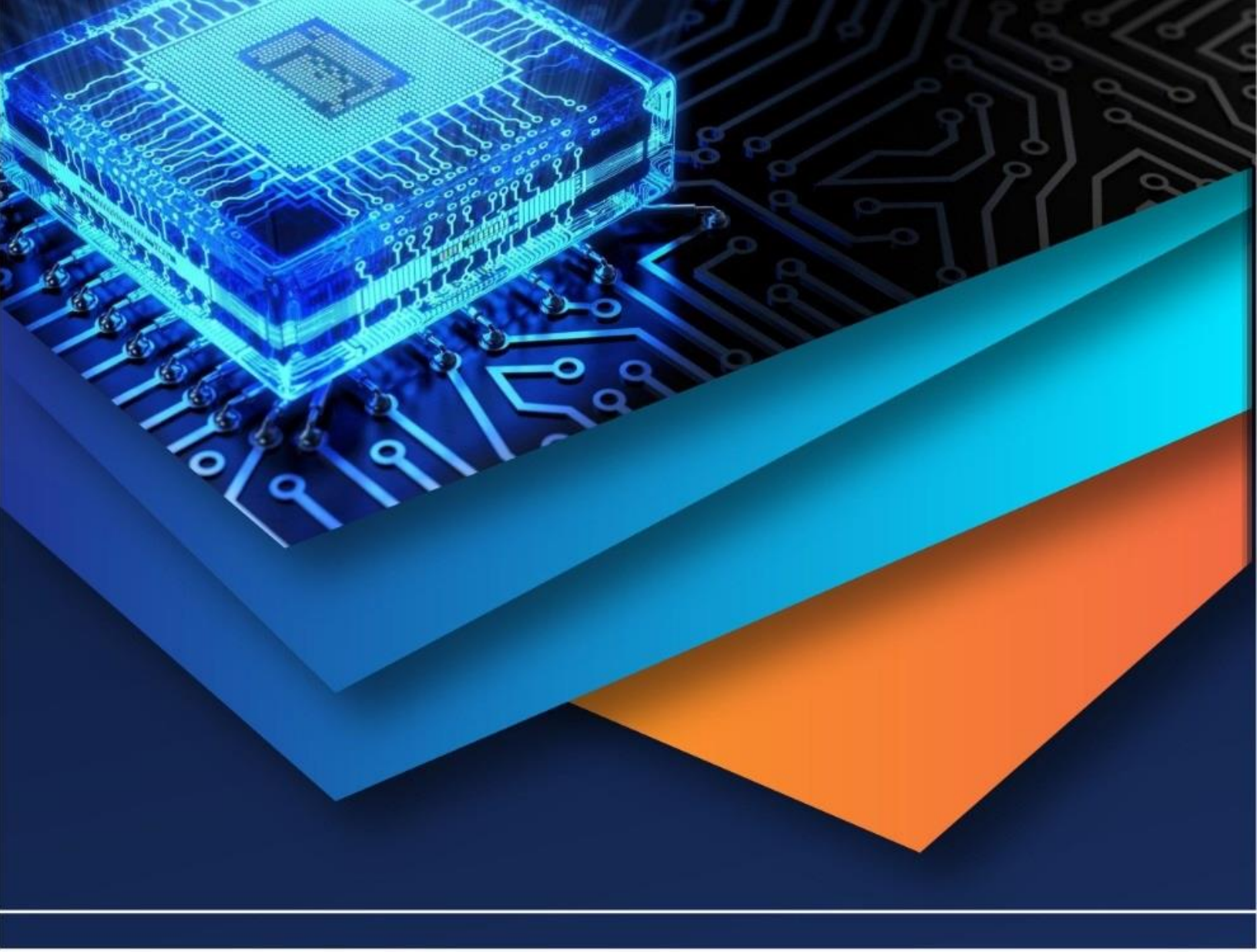


Fig 7.5 Result Page

REFERENCES

[1] J. Snell, K. Swersky, and R. Zemel, "Prototypical Networks for Few-Shot Learning," Advances in Neural Information Processing Systems (NeurIPS), vol. 30, pp. 4077–4087, 2017.
[2] A. Graves, A. R. Mohamed, and G. Hinton, "Speech Recognition with Deep Recurrent Neural Networks," IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP), pp. 6645–6649, 2013.
[3] T. Banerjee and V. Ramasubramanian, "Few-Shot Learning for Cross-Lingual Isolated Word Recognition in Indian Languages," ResearchGate Publication, 2021.
[4] S. Kumar and R. Singh, "Speech Recognition and Keyword Spotting using Deep Learning for Multilingual Indian Speech Data," International Journal of Computer Applications, vol. 178, no. 5, pp. 22–29, 2022.
[5] T. Javed, et al., "Keyword Spotting for Indian Languages using IndicSUPERB Benchmark," Indian Institute of Science, Bangalore, 2023.
[6] TensorFlow Developers, "TensorFlow: An End-to-End Open Source Machine Learning Platform," TensorFlow Documentation, 2024. [Online].
[7] Librosa Developers, "Librosa: Python Library for Audio and Music Analysis," Librosa Documentation, 2024. [Online].
[8] M. McFee et al., "librosa: Audio and Music Signal Analysis in Python," Proceedings of the 14th Python in Science Conference (SciPy), pp. 18–25, 2015.
[9] D. Jurafsky and J. H. Martin, Speech and Language Processing, 3rd ed., Pearson Education, 2023.
[10] F. Chollet, Deep Learning with Python, 2nd ed., Manning Publications, 2021.



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Call : 08813907089 (24*7 Support on Whatsapp)