Informe Final: Proyecto de Reconocimiento de Voz con ESP32 y TensorFlow Lite Micro

Resumen Ejecutivo

Este informe presenta el desarrollo y la implementación de un sistema de reconocimiento de voz en tiempo real basado en un microcontrolador ESP32-WROOM-32 y TensorFlow Lite Micro. El sistema permite entrenar un modelo con comandos de voz personalizados para controlar cuatro LEDs conectados a pines GPIO. Se cumplieron todos los requerimientos del cliente, incluyendo flexibilidad para reentrenar el modelo, operación sin conexión a internet y optimización para sistemas embebidos. Se utilizaron herramientas como Docker para entornos consistentes y Git/GitHub para versionado y colaboración.

1. Introducción

El propósito de este proyecto fue crear un sistema de reconocimiento de voz que permita al usuario entrenar un modelo con cuatro palabras personalizadas para controlar LEDs mediante un ESP32. El sistema debía ser eficiente, flexible y operar sin conexión a internet. Se empleó TensorFlow Lite Micro para optimizar el modelo y se implementó un modo de bajo consumo para reducir el uso de energía.

2. Componentes del Sistema

2.1 Hardware

- **Microcontrolador**: ESP32-WROOM-32
- **Micrófono**: INMP441 (interfaz I2S)
- **LEDs**: 4 LEDs en pines GPIO (12, 13, 14, 15)
- **Resistencias**: 220Ω por LED
- **Otros**: Cables y protoboard

2.2 Software

- **Scripts de Python**:
    - audios.py: Grabación de audios.

    - RedNeuronal_Español.py: Entrenamiento y exportación del modelo a .tflite.
- **Código para ESP32**:
    - main.cpp: Captura de audio, inferencia y control de LEDs.
- **Herramientas**: PlatformIO, Git, GitHub, Docker.

2.3 Modelo de Reconocimiento de Voz

- **Arquitectura**: CNN ligera (2 capas convolucionales, 1 capa densa).
- **Características**: MFCC (13 coeficientes, 40 frames/segundo).
- **Entrenamiento**: Data augmentation (ruido, pitch, velocidad).
- **Exportación**: Modelo quantizado a .tflite (<30 KB).

3. Desarrollo del Proyecto

3.1 Grabación y Entrenamiento

- **Grabación**: 20 muestras por palabra con audios.py.
- **Entrenamiento**: Extracción de MFCC, aumentación de datos y exportación a .tflite.
- **Optimización**: Quantización para el ESP32.

3.2 Implementación en el ESP32

- **Captura**: Configuración I2S para INMP441.
- **Inferencia**: TensorFlow Lite Micro.
- **Control**: Encendido de LEDs según comandos.
- **Bajo Consumo**: Modo de sueño profundo tras 5 segundos de inactividad.

3.3 Uso de Docker

Docker se empleó para encapsular dependencias (TensorFlow, Librosa) en un contenedor, asegurando un entorno de entrenamiento consistente y reproducible. Esto eliminó problemas de compatibilidad entre máquinas y facilitó la colaboración y entrega del proyecto. Un Dockerfile definió la imagen con Python 3.9 y las bibliotecas necesarias.

3.4 Uso de Git y GitHub

Git permitió versionar el código, mientras que GitHub alojó el repositorio con scripts, código del ESP32, modelo y documentación. Esto aseguró un acceso fácil para el cliente y colaboradores, con un README.md detallado para guiar el uso del sistema.

4. Pruebas y Resultados

- **Precisión**: >90% en pruebas.
- **Funcionamiento**: Respuesta correcta a comandos y control de LEDs.
- **Eficiencia**: Reducción de consumo con modo de sueño profundo.

5. Conclusiones y Recomendaciones

El proyecto cumplió todos los objetivos:

- Sistema flexible y personalizable.
- Operación offline.
- Optimización para bajo consumo.

Recomendaciones:

- Añadir más comandos.
- Mejorar el modo de bajo consumo.
- Explorar compresión adicional del modelo.

6. Entregables

- **Código**: Repositorio GitHub.
- **Modelo**: modelo_comandos_tflite.h en /src/.
- **Documentación**: README.md.

7. Agradecimientos

Gracias al cliente por su apoyo en este proyecto innovador.

Esp. Emmanuel Giudice
14 de mayo de 2025