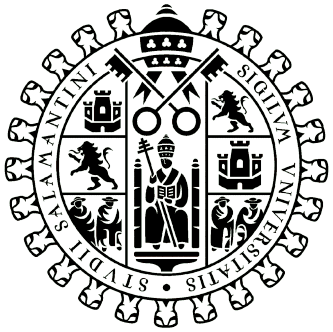
UNIVERSIDAD DE SALAMANCA

FACULTAD DE CIENCIAS



VNiVERSiDAD
D SALAMANCA

Desarrollo de una Aplicación de Reconocimiento de Voz.

DEVELOPMENT OF A SPEECH RECOGNITION APPLICATION.

Marta de Castro Leira

Abril 2025

Resumen:

En este informe se presenta el desarrollo de una aplicación sencilla de reconocimiento de voz, implementada en Python. El objetivo del proyecto es evaluar y comparar la precisión de diferentes APIs de reconocimiento de voz, incluyendo Google Speech-to-Text, Vosk y Whisper. La aplicación permite grabar y transcribir audio, ejecutar comandos de voz, traducir texto, detectar idioma y visualizar fonemas, todo a través de una interfaz gráfica desarrollada con Tkinter. El documento detalla la implementación, metodología y evaluación de los resultados obtenidos, así como futuras mejoras y aplicaciones de la herramienta desarrollada.

Abstract:

This report presents the development of a simple speech recognition application, implemented in Python. The goal of the project is to evaluate and compare the accuracy of different speech recognition APIs, including Google Speech-to-Text, Vosk and Whisper. The application allows recording and transcribing audio, executing voice commands, translating text, detecting language and visualizing phonemes, all through a graphical interface developed with Tkinter. The document details the implementation, methodology and evaluation of the results obtained, as well as future improvements and applications of the developed tool.

Índice

1. Introducción

El reconocimiento de voz es una tecnología fundamental en el campo de la inteligencia artificial y la interacción humano-computadora. Esta tecnología permite a los sistemas comprender y procesar el lenguaje hablado. El reconocimiento de voz es utilizado en una gran variedad de sectores, como la automatización de tareas, la mejora de la accesibilidad para personas con discapacidades, el desarrollo de asistentes virtuales, y la transcripción automática de discursos, entre otros.

En la última década, el avance de la inteligencia artificial y el aprendizaje automático ha potenciado la efectividad de las tecnologías de reconocimiento de voz. En la actualidad, plataformas como Siri, Alexa y Google Assistant son ejemplos claros del potencial de esta tecnología, que ya forma parte de la vida cotidiana de millones de personas. La adopción de la tecnología de reconocimiento de voz en aplicaciones de uso cotidiano ha revolucionado la manera en que los usuarios interactúan con sus dispositivos, brindando una experiencia mucho más fluida y natural.

Dentro de este contexto, el objetivo de este proyecto es desarrollar una aplicación basada en Python que permita grabar audio y transcribirlo de manera eficiente, utilizando tres APIs de reconocimiento de voz: Google Speech-to-Text, Vosk y Whisper. Estas tres opciones ofrecen diferentes enfoques y características, siendo de gran utilidad en distintos escenarios.

La aplicación desarrollada en este proyecto ha sido diseñada para proporcionar una interfaz de usuario intuitiva y accesible, que permita a los usuarios realizar transcripciones de audio de manera sencilla mediante los tres motores de reconocimiento de voz mencionados anteriormente. El sistema también ofrece varias funcionalidades adicionales como la conversión de texto a voz, análisis fonético y la capacidad de traducir las transcripciones a diferentes idiomas. Además, permite la ejecución de comandos de voz y la detección automática del idioma del audio, lo que optimiza la interacción y facilita el uso en distintos contextos.

Este proyecto está dividido en tres partes. En un principio se va a explicar la metodología utilizada para realizar la aplicación. Después, se analizará el desarrollo e implementación, donde se explicará más a fondo la aplicación creada. Por ultimo, se analizarán los resultados obtenidos realizando diversas pruebas en contextos variados con el fin de evaluar el rendimiento de los tres motores de reconocimiento de voz, y evaluando el funcionamiento del resto de funciones del sistema.

2. Metodología

Este sistema realiza la transcripción de audio a texto y permite cargar archivos de audio. Además, integra funcionalidades de síntesis de voz y análisis fonético, tales como la traducción a diferentes idiomas, transcripciones fonéticas, ejecución de comandos y detección automática de idioma.

En este apartado, se presenta la metodología utilizada en el desarrollo del sistema, buscando asegurar una implementación eficiente y robusta del sistema de reconocimiento de voz.

2.1. Implementación del Sistema

El sistema ha sido desarrollado utilizando el lenguaje de programación **Python**, con una interfaz gráfica construida sobre **Tkinter**. Para garantizar un procesamiento eficiente del audio y facilitar la interacción con el usuario, se han integrado diversas bibliotecas especializadas en el reconocimiento de voz, síntesis de texto a voz, manipulación de archivos de audio y traducción automática. A continuación, se describen las bibliotecas utilizadas y sus funciones principales.

PyAudio y **SpeechRecognition** permiten capturar y procesar audio en tiempo real desde el micrófono. En primer lugar, **PyAudio** facilita el acceso a los dispositivos de entrada y salida de audio, permitiendo capturar sonido del micrófono, reproducir audio y manipular flujos de audio de manera continua o por fragmentos. Sobre esta base, **SpeechRecognition** proporciona una interfaz sencilla para realizar transcripciones de voz a texto mediante diferentes motores de reconocimiento, como el motor en la nube de Google, además de ofrecer compatibilidad con otros motores de terceros.

Además de **SpeechRecognition**, se emplean dos bibliotecas adicionales para mejorar la precisión y versatilidad del reconocimiento de voz: **Whisper** de OpenAI y **Vosk**. Por un lado, **Whisper** destaca por su alta precisión en entornos ruidosos y su capacidad para transcribir audio en múltiples idiomas, siendo una excelente alternativa cuando el reconocimiento basado en la nube no es fiable. Por otro lado, **Vosk** permite realizar transcripción local sin necesidad de conexión a Internet, lo que lo convierte en una opción ideal para aplicaciones offline o con restricciones de red.

De manera inversa se utiliza **Pyttsx3**, una biblioteca que permite la conversión de texto a voz, y posibilita ajustar parámetros como el tono, la velocidad y el tipo de voz, lo que permite una mayor personalización de la experiencia auditiva proporcionando retroalimentación auditiva al usuario. **Pyttsx3** permite la personalización de la voz, el tono y la velocidad de la síntesis de voz.

Después de la grabación del audio, se utilizan las bibliotecas **Wave** y **NumPy** para la manipulación y análisis de archivos de audio. **Wave** se utiliza para leer, modificar y guardar archivos de audio en formato WAV, lo que permite trabajar con el contenido grabado. Por otro lado, **NumPy**

se utiliza para realizar análisis matemáticos y acústicos del audio. Esto incluye tareas como la reducción de ruido, ajuste de volumen y extracción de características acústicas que permiten mejorar la calidad del audio y optimizar la transcripción.

Otra biblioteca utilizada es **Deep_translator**, la cual es una herramienta que facilita la traducción automática de los textos transcritos a diferentes idiomas. Esta biblioteca integra múltiples motores de traducción para asegurar la mejor calidad en las traducciones. En conjunto con **Langdetect**, la cual detecta automáticamente el idioma del texto transcrito, se garantiza que el sistema seleccione el motor de traducción más adecuado.

Para análisis fonético y lingüístico, se utiliza la biblioteca **Phonemizer**, la cual es necesaria para la conversión de texto a fonemas, lo que es necesario para realizar análisis fonéticos del audio, como la mejora de pronunciación o la extracción de características lingüísticas. Este proceso es crucial para mejorar la calidad y precisión del reconocimiento de voz, especialmente en tareas relacionadas con la pronunciación o el aprendizaje de idiomas. Por tanto, esta biblioteca permite representar las palabras según su pronunciación en el Alfabeto Fonético Internacional (API).

Otra de las aplicaciones del sistema es el reconocimiento de audios de canciones. Para ello se utiliza la biblioteca **LyricsGenius**, la cual se emplea para la obtención de letras de canciones.

Por último, para gestionar configuraciones y almacenar datos de manera estructurada, se usa **Json5**. Esta biblioteca permite manejar configuraciones en formato JSON de forma más flexible, mejorando la personalización del sistema según las necesidades del usuario y asegurando una gestión eficaz de las configuraciones y preferencias.

Actualmente, el sistema ha sido desarrollado y probado en un entorno **macOS**, utilizando la versión **Python 3.8.6**, que ha demostrado ser la más estable para el funcionamiento de todas las dependencias utilizadas, incluyendo Whisper, Vosk y PyAudio.

2.2. Motores de transcripción de voz

Además de las bibliotecas ya mencionadas, el sistema emplea diferentes motores de transcripción de voz con el objetivo de obtener una solución más completa y adaptada a diversas situaciones. Estos motores se han seleccionado en función de su rendimiento, flexibilidad y capacidad para adaptarse a distintos escenarios. A continuación, se describen las tres principales opciones utilizadas en el sistema.

Uno de los motores de transcripción de voz utilizados es **Google Speech-to-Text (STT)**, el cual es uno de los motores de transcripción más conocidos y utilizados debido a su alta precisión y capacidad de transcripción en tiempo real. Su principal ventaja es su integración en la nube, ya

que permite un procesamiento potente y eficiente de grandes volúmenes de datos sin sobrecargar los recursos del dispositivo local. Además, debido a la infraestructura robusta de Google, este motor ofrece una gran escalabilidad y fiabilidad, siendo de gran utilidad en el caso de requerir un servicio de transcripción rápido y preciso en múltiples idiomas. La conectividad a la nube también proporciona acceso a actualizaciones frecuentes y mejoras en el modelo, lo que mantiene su rendimiento siempre actualizado en cuanto a las tecnologías de reconocimiento de voz.

Otro de los motores de transcripción de voz utilizados es **Vosk**, el cual es un motor de transcripción basado en modelos de código abierto. Su principal ventaja es que, al contrario que Google Speech-to-Text (STT), funciona de manera completamente local, lo que implica que no es necesario tener una conexión a Internet para realizar la transcripción. De esta manera, si la privacidad es necesaria o la conectividad a la red es limitada o inexistente, se trataría del motor más óptimo. Al ejecutarse en dispositivos locales, también reduce la latencia en la transcripción, lo cual es muy importante en aplicaciones en tiempo real. Además, en el caso de que un usuario necesite adaptar el sistema a unas necesidades específicas o desee integrar el motor en proyectos personalizados, Vosk le permite una gran flexibilidad y personalización ya que es un motor de código abierto.

Por último, se utiliza el motor de transcripción de vos **Whisper**, desarrollado por OpenAI, el cual es destacado por su capacidad para realizar transcripciones precisas incluso en entornos ruidosos. Este motor cuenta con modelos de aprendizaje profundo que le permiten entrenar con grandes cantidades de datos con el fin de mejorar constantemente la calidad de las transcripciones. Además, permite reconocer una amplia gama de acentos y variaciones del habla y es compatible con múltiples idiomas, lo que lo hace de gran utilidad en contextos multiculturales, internacionales o en situaciones donde la calidad del audio no es óptima. Debido a su capacidad para transcribir con precisión en ambientes ruidosos, se trata de un motor muy útil en tareas de transcripción de conversaciones realizadas en áreas públicas o en lugares con mucho ruido ambiental.

De esta manera, el sistema ha sido diseñado para ser flexible, permitiendo al usuario seleccionar el motor de transcripción más adecuado según las necesidades del contexto. Si se requiere un procesamiento en la nube y la precisión es la prioridad, **Google STT** es la mejor opción. Sin embargo, cuando no se tiene acceso a Internet o se necesita mantener la privacidad de los datos, **Vosk** es la opción más recomendable. Por otro lado, si se tuviera que realizar la transcripción en entornos con mucho ruido o se necesia un motor que se adapte bien a diversos acentos y lenguajes, habría que utilizar **Whisper**.

En resumen, la integración de estos tres motores de transcripción, junto con la flexibilidad para seleccionar el motor adecuado, proporciona una solución robusta, accesible y adaptable a

diversas necesidades.

3. Desarrollo

La aplicación fue desarrollada en **Python**, utilizando la biblioteca `Tkinter` para crear la interfaz gráfica de usuario. Su propósito principal es permitir la transcripción de voz a texto utilizando diferentes APIs de reconocimiento de voz, así como ofrecer funcionalidades adicionales como traducción, obtención de fonemas, identificación de idioma y ejecución de comandos por voz.

3.1. Flujo de Trabajo del Asistente de Voz

El flujo general de la aplicación sigue los siguientes pasos:

1. Captura de audio mediante grabación en vivo o carga de archivo.
2. Transcripción del audio mediante la API seleccionada (Google STT, Vosk o Whisper).
3. Presentación del texto transcrito en la interfaz.
4. Procesamiento adicional: traducción, conversión fonética, detección de idioma o ejecución de comandos.
5. Devolución del resultado en la interfaz, con opción de síntesis de voz.

3.2. Arquitectura General del Sistema

El sistema de reconocimiento de voz ha sido diseñado de manera modular, asegurando una implementación escalable y flexible. Está compuesto por los siguientes módulos principales:

- **Captura y Procesamiento de Audio**: Grabación en formato WAV con `pyaudio` y carga de archivos de audio externos.
- **Reconocimiento de Voz**: Transcripción automática del audio usando Google STT, Vosk y Whisper.
- **Interfaz Gráfica con Tkinter**: Interacción con el usuario mediante botones, cuadros de texto y menús desplegables.
- **Procesamiento de Texto**: Traducción, obtención de fonemas y detección de idioma.
- **Ejecución de Comandos de Voz**: Interpretación de comandos y ejecución de acciones en el sistema.
- **Integración con APIs Externas**: Búsqueda de canciones en Genius API y navegación web.

3.2.1. Interfaz Gráfica con Tkinter

La interfaz gráfica ha sido diseñada para permitir una interacción intuitiva. Está compuesta por un menú desplegable para seleccionar la API de reconocimiento de voz, diferentes botones para grabar, cargar archivos y transcribir audio, cuadros de texto con desplazamiento para mostrar transcripciones y resultados, y cuenta con opciones de traducción, detección de idioma y obtención de fonemas.

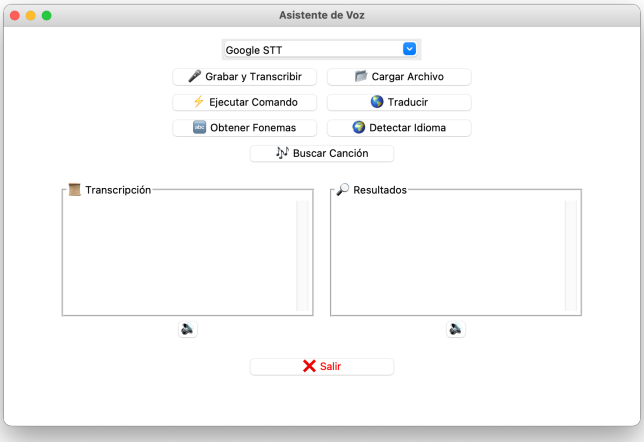


Figura 1: Vista general de la interfaz gráfica desarrollada con Tkinter.

3.2.2. Captura y Procesamiento de Audio

La aplicación permite al usuario grabar audio directamente desde el micrófono o cargar un archivo de audio ya existente desde el sistema de archivos. En ambos casos, el sistema se asegura de que el audio esté en el formato adecuado para que las APIs de reconocimiento de voz, especialmente `Vosk`, puedan procesarlo correctamente.

Para garantizar la compatibilidad, todos los audios procesados deben tener una frecuencia de muestreo de **16 kHz (16000 Hz)** y estar en **mono canal**. Esta configuración es especialmente importante para Vosk, ya que su modelo está entrenado para funcionar únicamente bajo estas condiciones. En caso contrario, la transcripción puede fallar o devolver resultados incorrectos.

Grabación de audio:

La grabación se activa mediante un botón que funciona como interruptor. Al pulsarlo por primera vez, se inicia la grabación y se indica visualmente que el sistema está “grabando”. Al pulsarlo de nuevo, se detiene la grabación, se procesa automáticamente el archivo de audio generado y se actualiza la interfaz con la transcripción correspondiente.

Esta funcionalidad se implementa utilizando un hilo independiente para capturar el audio de

forma continua hasta que el usuario detiene manualmente el proceso. La grabación se guarda como `audio_16k.wav` y posteriormente se transcribe utilizando el motor de reconocimiento seleccionado.

```
def grabar_audio_dinamico(nombre_archivo="audio_16k.wav"):
    formato = pyaudio.paInt16
    canales = 1
    tasa_muestreo = 16000
    tamano_bloque = 1024
    audio = pyaudio.PyAudio()
    frames = []

    stream = audio.open(format=formato, channels=canales, rate=
        tasa_muestreo, input=True, frames_per_buffer=tamano_bloque)

    while grabando:
        data = stream.read(tamano_bloque)
        frames.append(data)

    stream.stop_stream()
    stream.close()
    audio.terminate()

    with wave.open(nombre_archivo, 'wb') as wf:
        wf.setnchannels(canales)
        wf.setsampwidth(audio.get_sample_size(formato))
        wf.setframerate(tasa_muestreo)
        wf.writeframes(b''.join(frames))

    archivo = nombre_archivo
    api_seleccionada = api_var.get()
    texto = procesar_audio(api_seleccionada, archivo)

    root.after(0, lambda: finalizar_grabacion(texto))

def finalizar_grabacion(texto):
    actualizar_transcripcion(f"Transcripcion con {api_var.get()}", texto
        )
    hablar_con_idioma_detectado(text_output)
```

Carga de archivo de audio:

Si el usuario desea transcribir un archivo de audio ya existente (en formatos como `.wav`, `.mp3`, `.flac`, entre otros), el sistema realiza automáticamente una conversión para adaptarlo a las condiciones requeridas (16 kHz, mono). Esta conversión se lleva a cabo utilizando la biblioteca pydub y el resultado se guarda temporalmente como `audio_convertido.wav` en el directorio del proyecto.

```
def convertir_a_16k(audio_path):
    sonido = AudioSegment.from_file(audio_path)
    sonido = sonido.set_frame_rate(16000).set_channels(1)
    nuevo_path = "audio_convertido.wav"
    sonido.export(nuevo_path, format="wav")
    return nuevo_path


def cargar_audio():
    archivo = filedialog.askopenfilename(filetypes=[("Archivos de audio"
        , "*.wav *.mp3 *.flac *.ogg *.m4a")])
    if archivo:
        limpiar_texto()
        archivo_convertido = convertir_a_16k(archivo)
        api_seleccionada = api_var.get()
        texto = procesar_audio(api_seleccionada, archivo_convertido)
        actualizar_transcripcion(f"Transcripcion con {api_seleccionada}"
            , texto)
        hablar_texto(text_output)
```

Por tanto, todos los audios, tanto grabados como cargados, se procesan en formato **WAV, 16 kHz, mono**. Esto garantiza la compatibilidad con las APIs de reconocimiento utilizadas, especialmente con Vosk.

3.2.3. Transcripción Automática del Audio

La aplicación permite al usuario seleccionar entre tres motores de reconocimiento de voz para transcribir el audio capturado: **Google Speech-to-Text**, **Vosk** y **Whisper (OpenAI)**. Además, se ofrece una opción de *comparación*, que ejecuta simultáneamente los tres motores y muestra sus respectivas transcripciones, facilitando la evaluación comparativa de su rendimiento.

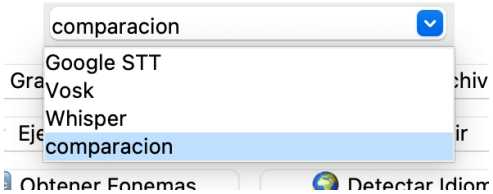


Figura 2: Opciones de selección del motor de reconocimiento de voz.

```
def procesar_audio(api, nombre_archivo="audio_16k.wav"):
    resultados = {}

    idioma_detectado = "es"
    try:
```

```
        model_temp = whisper.load_model("tiny")
        result_temp = model_temp.transcribe(nombre_archivo)
        texto_temp = result_temp["text"]
        idioma_detectado = detect(texto_temp)
        print(f"[DEBUG] Idioma detectado: {idioma_detectado}")
    except:
        pass

    # ------------------ GOOGLE STT ------------------
    if api in ["Google STT", "comparacion"]:
        recognizer = sr.Recognizer()
        try:
            with sr.AudioFile(nombre_archivo) as source:
                audio = recognizer.record(source)

            google_idioma = idioma_detectado + "-" + idioma_detectado.
                upper()

            resultados["Google STT"] = recognizer.recognize_google(audio
                , language=google_idioma)
        except sr.UnknownValueError:
            resultados["Google STT"] = "No se pudo entender el audio."
        except sr.RequestError as e:
            resultados["Google STT"] = f"Error con Google STT: {e}"

    # ------------------ VOSK ------------------
    if api in ["Vosk", "comparacion"]:
        model_path = "vosk-model-small-es-0.42"
        if os.path.exists(model_path):
            model = vosk.Model(model_path)
            recognizer = vosk.KaldiRecognizer(model, 16000)
            with wave.open(nombre_archivo, "rb") as wf:
                while True:
                    data = wf.readframes(4000)
                    if len(data) == 0:
                        break
                    recognizer.AcceptWaveform(data)
            result = json.loads(recognizer.FinalResult())
            resultados["Vosk"] = result.get("text", "No se pudo
                transcribir.")
        else:
            resultados["Vosk"] = "Modelo Vosk no encontrado."

    # ------------------ WHISPER ------------------
    if api in ["Whisper", "comparacion"]:
        model = whisper.load_model("base")
        result = model.transcribe(nombre_archivo)
```

```
        resultados["Whisper"] = result["text"]

    if api == "comparacion":
        return "\n\n".join([f"{k}:\n{v}" for k, v in resultados.items()
            ])
    else:
        return resultados.get(api, "API no soportada.")
```

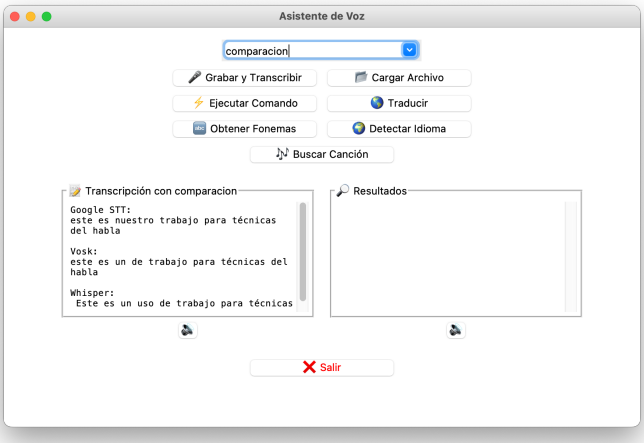


Figura 3: Ejemplo de transcripción de audio.

3.2.4. Procesamiento de Comandos de Voz

El sistema implementa un procesamiento básico de lenguaje natural para interpretar comandos simples dictados por el usuario a través del micrófono. Una vez que el audio se transcribe a texto, este se analiza mediante la función `procesar_comando()`, que compara el texto con un conjunto de comandos predefinidos y ejecuta la acción correspondiente en el sistema operativo o en el navegador web.

Para facilitar el reconocimiento, el texto se convierte primero a minúsculas. Luego, se comprueba si contiene alguna palabra clave asociada a cada comando en un diccionario. Si se encuentra una coincidencia, se ejecuta la acción programada.

A continuación se muestra un fragmento del código fuente:

```
def procesar_comando(texto):
    texto = texto.lower()

    comandos = {
        "abrir navegador": ["abrir navegador", "abrir chrome", "abrir
            Firefox", "abrir safari"],
        "que hora es": ["que hora es", "dime la hora", "hora actual"],
```

```
    "buscar en google": ["buscar en google", "buscar"],
    "poner musica en youtube": ["poner en youtube", "reproducir en
        youtube", "abrir youtube"],
    "realizar calculo": ["cuanto es", "calcular"],
}

resultado = "Comando no reconocido. Prueba con otro."

for comando, palabras_clave in comandos.items():
    if any(palabra in texto for palabra in palabras_clave):

        # Abrir navegador
        if comando == "abrir navegador":
            resultado = "Abriendo el navegador..."
            webbrowser.open("https://www.google.com")

        # Decir la hora actual
        elif comando == "que hora es":
            hora = datetime.now().strftime("%H:%M")
            resultado = f"Son las {hora}"

        #Buscar en Google
        elif comando == "buscar en google":
            busqueda = texto.replace("buscar en google", "").strip()
            if busqueda:
                resultado = f"Buscando '{busqueda}' en Google..."
                webbrowser.open(f"https://www.google.com/search?q={
                    busqueda}")

        #Poner musica en YouTube
        elif comando == "poner musica en youtube":
            cancion = texto.replace("poner en youtube", "").strip()
            if cancion:
                resultado = f"Reproduciendo '{cancion}' en YouTube
                    ..."
                webbrowser.open(f"https://www.youtube.com/results?
                    search_query={cancion}")

        #Realizar calculos matematicos
        elif comando == "realizar calculo":
            operacion = texto.replace("cuanto es", "").strip()
            try:
                resultado_calculo = eval(operacion)
                resultado = f"Resultado: {resultado_calculo}"
            except:
                resultado = "No pude calcular eso."
```

```
    return resultado
```

Cada comando está vinculado a una acción específica:

- **Abrir navegador**: lanza el navegador predeterminado con la página principal de Google.
- **Decir la hora**: obtiene la hora actual del sistema y la muestra en pantalla.
- **Buscar en Google**: realiza una búsqueda personalizada con las palabras clave dictadas.
- **Reproducir en YouTube**: abre los resultados en YouTube con el término proporcionado.
- **Cálculos matemáticos**: evalúa operaciones aritméticas sencillas como "cuánto es 5 + 3".

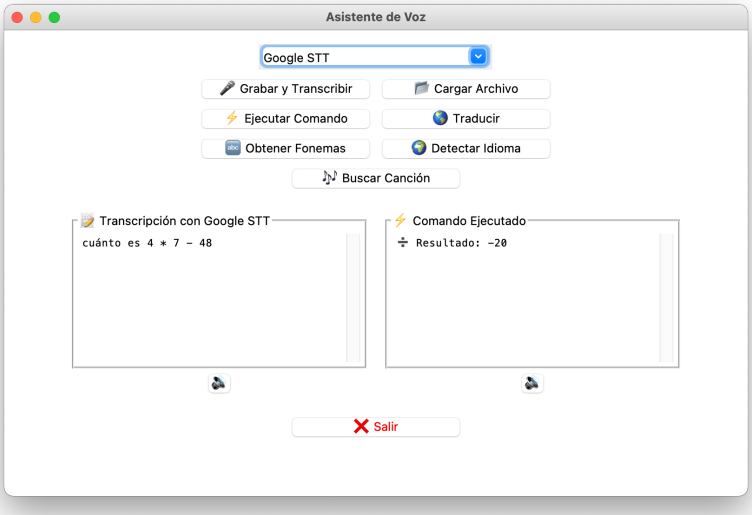


Figura 4: Ejemplo de ejecución del comando "cuánto es 4 * 7 - 48" mediante voz.

3.2.5. Procesamiento Adicional del Texto: Traducción, Fonética, Detección de Idioma y Búsqueda de Canciones

Además del reconocimiento y transcripción de voz, la aplicación ofrece una serie de funcionalidades adicionales para enriquecer el procesamiento del texto obtenido. Estas herramientas permiten traducir el contenido, obtener su transcripción fonética en formato IPA, detectar automáticamente el idioma del texto y buscar canciones a partir de fragmentos de letra. Además, los textos resultantes pueden ser reproducidos en voz alta en el idioma correspondiente gracias a la detección automática del idioma, proporcionando una experiencia más natural para el usuario.

Traducción del texto

El usuario puede seleccionar un idioma de destino desde un menú desplegable de la interfaz. Una vez traducido el contenido, este se muestra en la caja de resultados y es leído en voz alta en

el idioma traducido, utilizando una voz adaptada según disponibilidad del sistema.

Para mejorar la experiencia de usuario, al pulsar el botón “Traducir”, se despliega dinámicamente el menú de idiomas.

Figura 5: Interfaz menú para la selección de idioma.

Tras seleccionar el idioma y confirmar, se ejecuta la función `traducir_texto()`:

```
def traducir_texto():
    texto = text_output.get("1.0", tk.END).strip()
    idioma_destino = idioma_var.get()

    if not texto:
        messagebox.showwarning("Atencion","No hay texto para traducir.")
        return
    try:
        traduccion = GoogleTranslator(source="auto", target=
            idioma_destino).translate(texto)
        actualizar_respuesta(f"Traduccion ({idioma_dict[idioma_destino
            ]})", traduccion)
        frame_idioma.pack_forget()
        hablar_texto(respuesta_output, idioma=idioma_destino)
    except Exception as e:
        actualizar_respuesta("Error de Traduccion", f"Error: {e}")
```

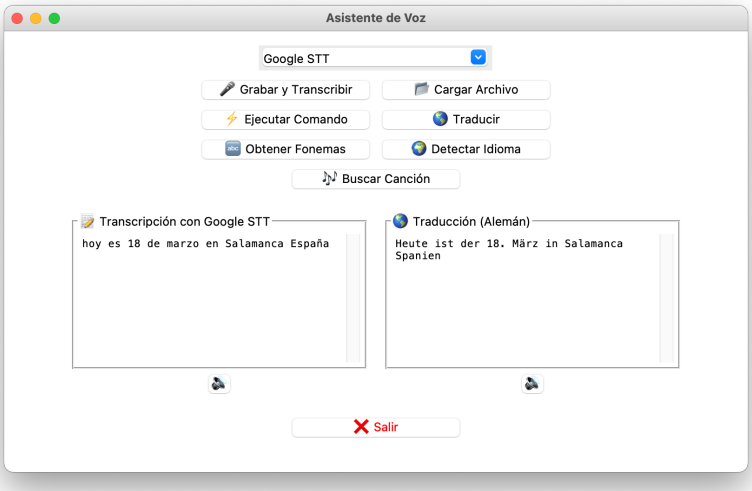


Figura 6: Interfaz de la traducción automática con selección de idioma.

Síntesis de voz

La aplicación cuenta con una funcionalidad de lectura en voz alta del contenido textual, implementada a través de la función `hablar_texto()`. Esta función utiliza la biblioteca `pyttsx3` para generar síntesis de voz a partir del texto presente en la interfaz. Se ajusta dinámicamente la velocidad de lectura, y si el idioma del texto no es español, se intenta seleccionar una voz compatible con el idioma detectado.

Para mejorar la experiencia, se incluye `hablar_con_idioma_detectado()`, que aplica detección automática del idioma del texto utilizando la biblioteca `langdetect`. De este modo, el sistema selecciona la voz más adecuada para leer el contenido con el acento correspondiente. En caso de no poder detectar el idioma, se utiliza la voz predeterminada en español.

```
def hablar_texto(text_widget, idioma="es"):
    texto = text_widget.get("1.0", tk.END).strip()
    if texto:
        engine = pyttsx3.init()
        engine.setProperty("rate", 130)

        if idioma != "es":
            voces = engine.getProperty('voices')
            for voz in voces:
                if idioma in voz.id:
                    engine.setProperty('voice', voz.id)
                    break

        engine.say(texto)
        engine.runAndWait()

def hablar_con_idioma_detectado(widget):
    texto = widget.get("1.0", "end").strip()
    if not texto:
        messagebox.showwarning("Atencion", "No hay texto para reproducir
            .")
        return
    try:
        idioma = detect(texto)
        hablar_texto(widget, idioma=idioma)
    except:
        hablar_texto(widget, idioma="es")
```

Gracias a esta integración, al pulsar el botón de altavoz junto a la transcripción o la traducción, el sistema vocaliza automáticamente el contenido adaptando el idioma y la entonación. Esto proporciona una lectura más natural y comprensible, especialmente útil para usuarios multilingües o en entornos de aprendizaje de idiomas.

Obtención de fonemas

El sistema incluye una funcionalidad que permite al usuario obtener la representación fonética

del texto ingresado. Esta opción se activa mediante el botón *"Obtener Fonemas"*, que ejecuta la función `obtener_fonemas()`.

Esta función utiliza la biblioteca `phonemizer` junto con el backend `espeak` para convertir el texto escrito en su correspondiente transcripción fonética, teniendo en cuenta el idioma detectado (por defecto, español). Además, incluye una gestión básica de errores para manejar entradas vacías o inválidas. A continuación, se muestra el código correspondiente:

```
def obtener_fonemas(texto, idioma_detectado="es"):
    try:
        fonemas = phonemize(texto, language=idioma_detectado, backend="
            espeak")
        return fonemas
    except Exception as e:
        return f"Error al obtener fonemas: {e}"
```

Esta función es especialmente útil para tareas de análisis lingüístico ya que proporciona una forma automatizada de conocer cómo se pronuncian las palabras ingresadas.

Detección automática de idioma

Esta funcionalidad se basa en la biblioteca `langdetect`, que analiza el texto y devuelve el código del idioma predominante. Una vez detectado el idioma, se traduce el código ISO a un nombre legible para el usuario (como "Español" o "Inglés") mediante un diccionario auxiliar. El resultado se muestra automáticamente en la interfaz, y puede ser leído en voz alta en el idioma detectado:

```
def detectar_idioma(texto):
    if not texto.strip():
        actualizar_respuesta("Error", "No hay texto para detectar idioma
            .")
        return

    try:
        idioma = detect(texto)
        nombres_idiomas = {
            "en": "Ingles", "es": "Espanol", "fr": "Frances",
            "de": "Aleman", "it": "Italiano", "pt": "Portugues"
        }
        idioma_nombre = nombres_idiomas.get(idioma, "Desconocido")
        actualizar_respuesta("Idioma Detectado", idioma_nombre)
    except Exception as e:
        actualizar_respuesta("Error", f"No se pudo detectar el idioma: {
            e}")
```

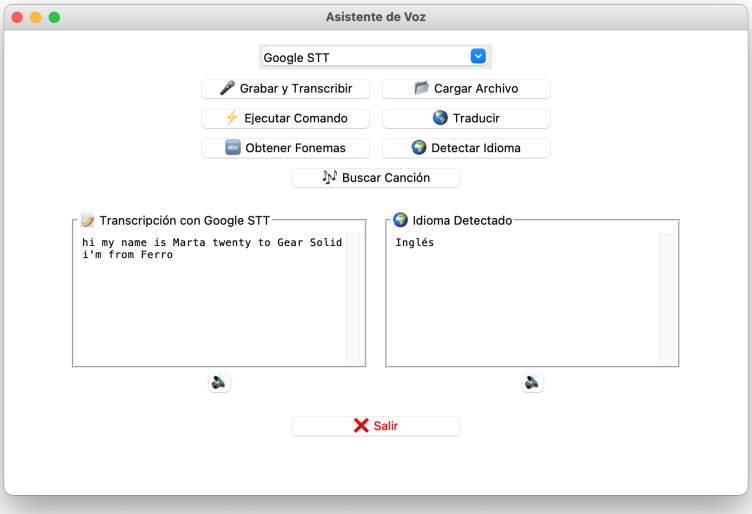


Figura 7: Detección automática del idioma del texto.

Búsqueda de canciones mediante Genius API

La aplicación permite buscar canciones a partir de fragmentos de letra gracias a la integración con la API de Genius. El sistema realiza una petición al endpoint de búsqueda de letras y, si encuentra resultados, muestra el título y el artista de la primera coincidencia.

El funcionamiento está encapsulado en la función `buscar_cancion()`, que incluye manejo de errores si no se obtiene respuesta válida:

```
def buscar_cancion(letra):
    try:
        resultados = genius.search_lyrics(letra)
        if resultados["sections"]:
            cancion = resultados["sections"][0]["hits"][0]["result"]
            return f"Cancion: {cancion['title']} - {cancion['
                primary_artist']['name']}"
        return "No se encontro la cancion."
    except Exception as e:
        return f"Error en la busqueda de canciun: {e}"
```

Figura 8: Resultado de búsqueda de una canción con fragmentos de letra.

En conjunto, estas funcionalidades amplían notablemente las capacidades del asistente de voz, añadiendo herramientas lingüísticas, culturales y musicales que enriquecen la interacción del usuario más allá de la simple transcripción de voz a texto.

3.3. Ejecución automática

Actualmente, la aplicación ha sido desarrollada y probada en **macOS**, donde funciona correctamente. Para facilitar su uso, se ha implementado un script de instalación y ejecución automática llamado `setup.sh`, que se encarga de crear el entorno virtual y ejecutar el asistente con todas las dependencias correctamente configuradas.

```
#!/bin/bash

echo "Creando␣entorno␣virtual..."
python3 -m venv asistente_env

echo "Activando␣entorno␣virtual..."
source asistente_env/bin/activate

echo "Actualizando␣pip..."
pip install --upgrade pip

echo "Instalando␣dependencias..."
pip install tk pyaudio speechrecognition vosk pyttsx3 pydub deep-
    translator langdetect phonemizer lyricsgenius torch openai-whisper

echo "Fijando␣numpy<2␣por␣compatibilidad..."
pip install "numpy<2"
```

```
echo "Instalacion completada."
echo "Ejecutando el asistente de voz..."
python reconocimiento.py
```

4. Resultados y Comparación

Para evaluar el rendimiento de los tres motores de transcripción utilizados en la aplicación (Google STT, Vosk y Whisper), se han realizado cinco pruebas en distintos contextos que simulan escenarios de uso real.

Con el fin de evaluar su rendimiento en diferentes entornos, los cinco contextos en los que se han tenido en cuenta son un entorno silencioso, uno con ruido de fondo, otro sin conexión a internet, una frase compleja de carácter técnico y una frase en inglés. Los resultados en dichas situaciones se presentan en las siguientes tablas, en las que se puede observar el texto transcrito por cada motor.

4.1. Entorno silencioso

En un entorno sin ruido, los tres motores de transcripción ofrecen resultados precisos y comprensibles. Google Speech-to-Text (STT) y Vosk captan correctamente la estructura general de la frase, aunque Google STT recorta la transcripción antes de completarla. Este motor también es el que menos cantidad de palabras reconoce en esta prueba.

Por su parte, Whisper realiza una transcripción completa y más pulida, incorporando elementos de puntuación como comas y mayúsculas al inicio de la frase. Además, añade puntos suspensivos al final, interpretando que el usuario seguía hablando cuando la grabación se interrumpió, lo que demuestra una mayor sensibilidad contextual. Estos detalles hacen que el texto generado por Whisper resulte más natural y fluido.

Fuente	**Texto**
Lo que se dijo	Hoy hace un día nublado para salir a la calle, así que creo que me quedo en casa.
Google STT	hoy hace un día nublado para salir a la calle así que creo que
Vosk	hoy hace un día nublado para salir a la calle así que creo que me quedo
Whisper	Hoy hace un día nublado para salir a la calle, así que creo que me quedó...

Cuadro 1: Comparación de transcripciones en entorno silencioso.

4.2. Con ruido de fondo

En presencia de ruido ambiental generado por varias personas hablando simultáneamente, el rendimiento de los motores de transcripción presenta diferencias significativas. Google Speech-to-Text (STT) logra mantener una transcripción coherente y comprensible, identificando correctamente la frase pronunciada por el usuario a pesar de las interferencias.

En contraste, Vosk muestra errores sintácticos notables, introduciendo palabras que no fueron pronunciadas y alterando el significado de la frase, lo cual indica una mayor susceptibilidad al ruido de fondo.

Whisper, por su parte, ofrece una transcripción completamente distorsionada e ininteligible. Esto sugiere que, aunque es potente en condiciones limpias, su rendimiento se ve fuertemente afectado cuando el audio contiene múltiples voces o interferencias. Este comportamiento se debe a la sensibilidad del modelo a la calidad acústica de la entrada.

En conjunto, esta prueba muestra que Google STT es más robusto ante ruido ambiental leve, mientras que Whisper y Vosk requieren condiciones acústicas más controladas para garantizar una buena precisión.

Fuente	**Texto**
Lo que se dijo	Érase una vez un barquito triste y oscuro.
Google STT	érase una vez un barquito triste y oscuro
Vosk	era así al principio me del viento triste y oscuro
Whisper	Era si almenyszumenibrant trista hi os cudo?

Cuadro 2: Comparación de transcripciones con ruido de fondo.

4.3. Sin conexión a Internet

Esta prueba evidencia una de las principales limitaciones de Google Speech-to-Text (STT): su dependencia de conexión a Internet. Al no disponer de acceso a la red, el sistema no puede enviar el audio al servidor de Google para su procesamiento y lanza un mensaje de error, quedando completamente inutilizado.

Por el contrario, tanto Vosk como Whisper son motores que operan de forma local, sin necesidad de conexión, lo que les permite funcionar correctamente en entornos sin acceso a Internet. En esta prueba, ambos motores fueron capaces de transcribir la frase con precisión.

Además, Whisper destaca por ofrecer una transcripción con puntuación adecuada, utilizando mayúsculas y cifras correctamente, lo que contribuye a una mejor presentación del resultado. Este

comportamiento hace que tanto Vosk como Whisper sean opciones recomendables en contextos donde se prioriza la privacidad o se carece de conectividad.

Fuente	Texto
Lo que se dijo	Hoy es miércoles 19 de marzo a las 10:11 de la mañana.
Google STT	Error con Google STT: recognition connection failed
Vosk	hoy es miércoles diecinueve de marzo a las diez y once de la mañana
Whisper	Hoy es miércoles 19 de marzo a las 10 y 11 de la mañana.

Cuadro 3: Comparación de transcripciones sin conexión a Internet.

4.4. Frase técnica compleja

Cuando se introduce una frase con contenido técnico, los motores de transcripción muestran diferencias importantes en su capacidad para manejar vocabulario especializado. Google Speech-to-Text (STT) destaca por su precisión, reconociendo correctamente expresiones como "acelerador de partículas" y el nombre propio "CERN". Esta correcta identificación sugiere que su modelo ha sido entrenado con grandes volúmenes de datos de naturaleza técnica y científica.

Vosk, por su parte, presenta una omisión significativa al no detectar la palabra "CERN" y ofrecer en su lugar una frase gramaticalmente inconsistente. Este comportamiento puede deberse a su entrenamiento con un conjunto de datos más limitado, especialmente en cuanto a términos científicos o específicos.

Whisper comete un error fonético al interpretar "CERN" como "cerne", aunque mantiene la estructura general de la frase. A pesar del fallo en la palabra clave, se observa que Whisper conserva la puntuación y la coherencia textual, lo que indica un modelo más robusto en cuanto a la presentación del texto.

Estos resultados evidencian que Google STT es actualmente el motor más fiable en contextos técnicos, seguido de cerca por Whisper en términos de legibilidad, mientras que Vosk puede verse limitado en este tipo de dominios específicos.

Fuente	Texto
Lo que se dijo	El experimento se realizó en el acelerador de partículas del CERN de Ginebra.
Google STT	el experimento se realizó en el acelerador de partículas del CERN de Ginebra
Vosk	el experimento se realizó en el acelerador de partículas del de ginebra
Whisper	El experimento se realizó en el acelerador de partículas del cerne de ginebra.

Cuadro 4: Comparación de transcripciones con frase técnica.

4.5. Frase en inglés

En esta prueba se evalúa la capacidad de los motores para transcribir correctamente una frase hablada en inglés. Tanto Google Speech-to-Text (STT) como Whisper muestran un rendimiento sobresaliente, generando transcripciones completas, gramaticalmente correctas y coherentes con el mensaje original. Estos resultados evidencian su soporte multilingüe y su capacidad de adaptación automática al idioma detectado, sin necesidad de configuración adicional.

En contraste, Vosk produce una transcripción ininteligible y sin sentido, lo que indica que el modelo cargado está exclusivamente entrenado para el idioma español. A diferencia de Whisper o Google STT, Vosk requiere explícitamente el uso de un modelo entrenado para cada idioma específico y no realiza detección automática de idioma.

Este experimento demuestra que, en aplicaciones donde se espera trabajar con múltiples lenguas, Whisper y Google STT son soluciones más versátiles. Vosk, aunque funcional en contextos monolingües y sin conexión, no es adecuado para entornos multilingües sin una configuración previa del modelo correspondiente.

Fuente	Texto
Lo que se dijo	Hi, my name is Marta and this is a test to see if this works.
Google STT	hi my name is Marta and this is to see it this works
Vosk	mis marta han dije testuz y de su wax
Whisper	Hi, my name is Marta and this is a test to see if this works.

Cuadro 5: Comparación de transcripciones en idioma inglés.

En conjunto, las pruebas demuestran que cada motor de transcripción tiene ventajas y limitaciones específicas, y su elección dependerá del contexto de uso: si se requiere precisión técnica,

robustez frente al ruido, o funcionalidad offline.

Además de la evaluación de los motores de transcripción, también se verificaron otras funcionalidades complementarias de la aplicación, obteniendo resultados correctos en todos los casos:

- **La detección automática de idioma** mediante `langdetect` funcionó adecuadamente, identificando correctamente el idioma predominante del texto transcrito en la mayoría de los casos. No obstante, se debe tener en cuenta que esta detección es más precisa cuando se proporciona una cantidad suficiente de texto. En frases muy cortas o palabras sueltas, el modelo puede confundirse con idiomas similares, lo que podría afectar a funcionalidades dependientes del idioma como la síntesis de voz.
- **La traducción de textos** con `deep-translator` a distintos idiomas fue precisa y se integró correctamente en la interfaz, incluyendo la lectura en voz alta del resultado traducido con la voz adaptada al idioma de destino.
- **La transcripción fonética** con `phonemizer` devolvió representaciones correctas en formato IPA para los textos analizados, ajustando dinámicamente el idioma de fonetización al idioma detectado automáticamente.

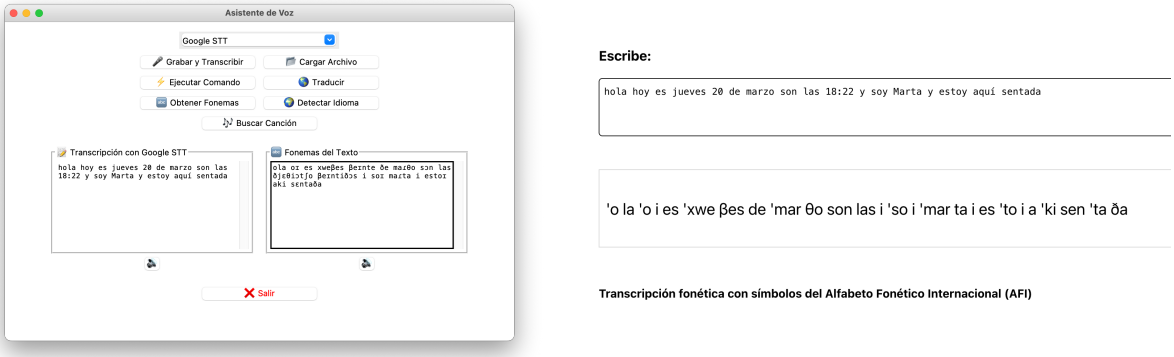


Figura 9: Comparación de resultados de transcripción fonética.

- **La búsqueda de canciones** a partir de fragmentos de letra mediante la Genius API permitió recuperar correctamente información sobre el título y el autor en los casos probados, mostrando la utilidad de la integración externa. No obstante, si no se encuentra ninguna coincidencia, el sistema lanza un error tipo `list index out of range`, lo cual indica que la estructura esperada en la respuesta no está presente. Para mejorar la robustez del sistema, sería recomendable capturar este caso con una gestión de errores más específica y mostrar un mensaje más claro al usuario, como "No se encontró ninguna canción relacionada".

Estas funcionalidades adicionales complementan el sistema de transcripción y enriquecen la experiencia del usuario, proporcionando herramientas útiles tanto para el análisis lingüístico como para la interacción natural con el asistente.

5. Conclusiones y Líneas Futuras

Conclusiones

A partir de los resultados obtenidos durante las pruebas, se pueden extraer varias conclusiones relevantes respecto al comportamiento y utilidad de los distintos motores de transcripción de voz integrados en la aplicación:

- **Google Speech-to-Text (STT)** ofrece una alta precisión en condiciones ideales y es especialmente eficaz en entornos ruidosos o con contenido técnico. Sin embargo, su principal limitación es la dependencia de una conexión a Internet, lo que puede restringir su uso en contextos offline o con problemas de conectividad.
- **Vosk** destaca por su funcionamiento completamente local, lo que le permite transcribir sin necesidad de acceso a Internet y mantener la privacidad del usuario. Aun así, presenta limitaciones en precisión frente a frases complejas, entornos con ruido y, especialmente, en idiomas distintos al configurado, al no contar con detección automática de idioma.
- **Whisper**, desarrollado por OpenAI, combina lo mejor de ambos mundos: funciona sin conexión, soporta múltiples idiomas y ofrece una transcripción pulida y bien estructurada. Si bien puede verse afectado por ruido de fondo, su desempeño en frases técnicas y en inglés lo posiciona como el motor más versátil de los tres evaluados.

Además, se comprobó que las funcionalidades adicionales como la detección automática de idioma, la traducción de texto, la obtención de fonemas y la búsqueda de canciones funcionaron correctamente, añadiendo valor al sistema y mejorando la interacción con el usuario.

Líneas Futuras

Este proyecto sienta las bases para realizar una aplicación de reconocimiento de voz. Sin embargo, existen posibles mejoras y ampliaciones que se podrían realizar. Algunas posibles líneas de trabajo incluyen mejorar la robustez en entornos con ruido, es decir, aplicar técnicas de preprocesamiento y reducción de ruido con el fin de aumentar la calidad del audio antes de la transcripción. Además, se podría automatizar la selección del motor de transcripción mediante la detección del entorno, idioma o tipo de frase. Esto permitiría una respuesta más inteligente del sistema según las condiciones del usuario.

Otra posible mejora sería integrar un sistema de entrenamiento personalizado, que permita adaptar el reconocimiento de voz a la voz específica del usuario, mejorando progresivamente la precisión en cada uso. También se podrían incorporar funcionalidades de almacenamiento de transcripciones y comandos, lo que permitiría registrar el historial de interacciones y realizar análisis posteriores en contextos como asistentes personales, educación o accesibilidad.

Dado que actualmente la aplicación ha sido desarrollada y probada únicamente en **macOS**, sería interesante estudiar su portabilidad a otros sistemas operativos como **Linux** o **Windows**. En este sentido, sería necesario adaptar el proceso de instalación y ejecución, por ejemplo, creando un script equivalente al actual `setup.sh` para entornos Windows (como un `setup.bat`), y ajustando las dependencias del sistema en distribuciones Linux, especialmente aquellas relacionadas con audio.

Además, como la aplicación ha demostrado ser estable en la versión **Python 3.8.6**, se recomienda mantener dicha versión en futuras instalaciones para evitar problemas de compatibilidad, especialmente con bibliotecas como Whisper o PyAudio. También sería útil documentar con mayor detalle el proceso de instalación y uso en diferentes plataformas, e incluso plantearse la automatización de pruebas multiplataforma mediante máquinas virtuales, para garantizar su correcto funcionamiento en cualquier entorno.

Existen muchas otras líneas de trabajo futuras en este ámbito. Estas mejoras contribuirían a convertir la aplicación en una herramienta aún más potente, flexible y accesible para una amplia gama de usuarios y escenarios de uso.

Referencias

[1] Apple. (n.d.). *Siri*. Apple. Recuperado de https://www.apple.com/es/siri/

[2] Amazon. (n.d.). *Alexa Developer*. Amazon. Recuperado de https://developer.amazon.com/es-ES/alexa

[3] Google. (n.d.). *Asistente de Google*. Google. Recuperado de https://assistant.google.com/intl/es_es/

[4] López, X. (s.f.). *Transcriptor Fonético*. Recuperado el 19 de marzo de 2025, de https://xavierlopez.dev/transcriptorfonetico/

[5] Genius. (n.d.). *Genius API Documentation*. Genius. Recuperado de https://docs.genius.com/

[6] Alpha Cephei. (n.d.). *Vosk Speech Recognition Toolkit – Models*. Vosk. Recuperado de https://alphacephei.com/vosk/models

[7] Google Cloud. (n.d.). *Speech-to-Text*. Google Cloud. Recuperado de https://cloud.google.com/speech-to-text?hl=es

[8] OpenAI. (2022). *Whisper: Open source neural speech recognition system*. OpenAI. Recuperado de https://openai.com/index/whisper/