Escuela Politécnica Superior de Linares

Grado en Ingeniería de Tecnologías de Telecomunicación

UNIVERSIDAD DE JAÉN

Escuela Politécnica Superior de Linares

Trabajo Fin de Grado

SERVICIO WEB PARA LA CONVERSIÓN DE CANCIONES A FORMATO KARAOKE USANDO INTELIGENCIA ARTIFICIAL

Alumno: Ramón Elbal Ruiz

Tutores: Pedro Vera Candeas

Pablo Antonio Cabañas Molero

Depto.: Ingeniería de Telecomunicación

Junio, 2024

UNIVERSIDAD DE JAÉN

Escuela Politécnica Superior de Linares

Grado en Ingeniería de Tecnologías de la Telecomunicación

Trabajo Fin de Grado

SERVICIO WEB PARA LA CONVERSIÓN DE CANCIONES A FORMATO KARAOKE USANDO INTELIGENCIA ARTIFICIAL

Alumno: Ramón Elbal Ruiz

Tutores: Pedro Vera Candeas

Pablo Antonio Cabañas Molero

Depto.: Ingeniería de Telecomunicación

ELBAL RUIZ RAMON - 26509331Z

Firmado digitalmente por ELBAL RUIZ RAMON - 26509331Z
Fecha: 2024.06.24 10:25:30 +02'00'

Firma del autor

CABAÑAS MOLERO PABLO ANTONIO - 77342099E

Firmado digitalmente por CABAÑAS MOLERO PABLO ANTONIO - 77342099E
Nombre de reconocimiento (DN): c=ES, serialNumber=IDCES-77342099E, givenName=PABLO ANTONIO, sn=CABAÑAS MOLERO, cn=CABAÑAS MOLERO PABLO ANTONIO - 77342099E
Fecha: 2024.06.24 11:57:44 +02'00'

VERA CANDEAS PEDRO - 26228400M

Digitally signed by VERA CANDEAS PEDRO - 26228400M
Date: 2024.06.24 11:41:04 +02'00'

Firma de los tutores

ÍNDICE GENERAL

Índice de figuras 6

1 Introducción 8

1.1 Justificación, contexto y descripción del TFG 8

1.2 Objetivos 9

2 Estado del arte 10

3 Materiales y métodos 13

3.1 Esquema del sistema 13

3.2 Lenguajes de programación 14

3.2.1 Back-end: Python 14

3.2.2 Front-end: JavaScript 14

3.3 Software externo al desarrollo de la aplicación 14

3.3.1 Visual Studio Code 14

3.3.2 GitHub 14

3.3.3 Postman 15

3.3.4 VirtualBox 15

3.3.5 UltraStar 15

3.4 Back-end (Aplicación de conversión) 15

3.4.1 Anaconda/Mamba 15

3.4.2 UltraSinger 16

3.4.3 FastAPI 16

3.4.4 Uvicorn 16

3.4.5 Tensorflow 16

3.4.6 ImageMagick: 17

3.4.7 Prosodic: 17

3.4.8 Pyverse: 17

3.4.9 Librosa: 18

3.4.10 Moviepy: 18

3.4.11 Modelos preentrenados de inteligencia artificial 18

3.5 Interfaz de usuario 26
3.5.1 React 26
3.5.2 Fetch....28
3.6 Explicación del TFG y sus componentes 28
3.6.1 Back-end....28
3.6.2 Interfaz de usuario 29
3.6.3 Servicio REST....29
3.7 Flujo de la aplicación 31
3.7.1 Recepción del POST:....33
3.7.2 Separación de voz e instrumental: 33
3.7.3 Generación de letra:....33
3.7.4 Proceso de cálculo de pitch: 33
3.7.5 Proceso de obtención de sílabas: 34
3.7.6 Proceso de asociación de sílabas a notas: 34
3.7.7 Proceso de generación del TXT de UltraStar: 34
3.7.8 Proceso de video: 35
4 Resultados 36
4.1 Comprobación del manejo correcto a nivel de usuario....36
4.1.1 Conversión de un archivo de audio a un archivo UltraStar....36
4.1.2 Comparación de la calidad del karaoke con uno existente....39
4.1.3 Conversión de un archivo de audio a un archivo de video 42
4.2 Comprobación del manejo correcto de situaciones especiales: 43
4.2.1 Archivo de audio no en formato mp3....43
4.2.2 Archivo que no es de audio....44
4.2.3 Archivo superior a 25 MB....44
4.2.4 El usuario abandona la página durante el proceso....44
4.2.5 Se recibe más de una solicitud a la vez 44
5 Conclusiones y líneas futuras de desarrollo....47
6 Anexos 49

6.1 Anexo I: Manual técnico y de despliegue 49
6.1.1 Requerimientos de despliegue 49
6.1.2 Instalar CUDA 49
6.1.3 Instalar las dependencias en un entorno virtual 49
6.1.4 Instalar pytorch 51
6.1.5 Despliegue del servidor 52
6.1.6 Solución de problemas 53
6.2 Anexo II: Formato y uso de archivos UltraStar 55
7 Referencias 58

ÍNDICE DE FIGURAS

Ilustración 1: Esquema de las interacciones entre sistemas de la aplicación.....13
Ilustración 2: Comparativa de WER de transcripciones de Whisper en función del idioma, modelo y dataset.....20
Ilustración 3: Comparativa de modelos de transcripción bajo diferentes corpus lingüísticos.....22
Ilustración 4: Diagrama del funcionamiento de WhisperX.....22
Ilustración 5: Desempeño de tres modelos pre-entrenados en una tarea de reconocimiento de habla.....24
Ilustración 6: Esquema de la arquitectura del modelo Hybrid Transformer Demucs 25
Ilustración 7: Comparativa de precisión de modelos de detección de pitch 26
Ilustración 8: Carpeta donde se almacena la aplicación web.....28
Ilustración 9: Diagrama de flujo del funcionamiento de la aplicación 32
Ilustración 10: Dispositivo a partir del cual se ha realizado la conexión a la interfaz desplegada.....36
Ilustración 11: Botón de subida de archivo deshabilitado 36
Ilustración 12: Resultado de la terminal donde se ha desplegado el servicio.....37
Ilustración 13: Notificación de navegador informando de una descarga de archivo. 37
Ilustración 14: Contenido del archivo comprimido.....38
Ilustración 15: Archivo de texto incluido en el UltraStar 38
Ilustración 16: Selector de canciones de UltraStar WorldParty 39
Ilustración 17: Imagen de una partida de UltraStar con el archivo recibido.....39
Ilustración 18: Canción elegida como referencia 40
Ilustración 19: Canción descargada de Ultrastar 40
Ilustración 20: Canción obtenida de la aplicación 41
Ilustración 21: Interfaz de usuario con la opción cambiada.....42
Ilustración 22: Video obtenido tras el proceso 42
Ilustración 23: Video obtenido siendo reproducido 43
Ilustración 24: Conversión iniciada desde múltiples ventanas, cada una con un archivo diferente 45
Ilustración 25: La primera conversión termina y la segunda se inicia justo después 45
Ilustración 26: Descargas realizadas con éxito.....46
Ilustración 27: Resultado del comando nvcc --version.....49

Ilustración 28: Página web que permite ver el comando necesario para instalar dependencias 52
Ilustración 29: Instalación de las dependencias de Pytorch 52
Ilustración 30: Despliegue de la aplicación de conversión 52
Ilustración 31: Navegador web mostrando la interfaz de usuario 53
Ilustración 32: Carpeta de instalación de ImageMagick 54
Ilustración 33: Archivo policy.xml 54
Ilustración 34: Directorio de canciones de UltraStar 56

1 INTRODUCCIÓN

En el siguiente punto se detalla el contexto y desarrollo del trabajo realizado, junto a la estructura de éste.

1.1 Justificación, contexto y descripción del TFG

Hoy en día, la inteligencia artificial se ha posicionado como una de las herramientas más extendidas y versátiles que existen. Su capacidad de generar contenido de manera eficiente, así como de automatizar tareas de análisis y extracción de datos de manera precisa, la hacen especialmente útil en una variedad de sectores.

Las aplicaciones web se sitúan como el entorno idóneo para hacer acercar estas herramientas para el público. Esto es debido a su facilidad de acceso desde cualquier dispositivo electrónico con capacidad de conectarse a internet mediante un navegador.

Por otra parte, el karaoke es un medio de entretenimiento que, si bien ha sido gravemente afectado por la pandemia, sigue manteniendo cierta popularidad, especialmente en Japón, el país en el que surgió. La mayoría de software de karaoke, como son *Ultrastar, Karafun* y *Singa*, se distribuye como aplicaciones de escritorio, tanto como código abierto como propietario. Sin embargo, el auge de los teléfonos móviles y demás dispositivos electrónicos ha llevado a estas aplicaciones a desarrollarse en múltiples dispositivos para mejorar su disponibilidad y ofrecer funcionalidades que las distingan de otros servicios. Esta tendencia también ha llevado al desarrollo de aplicaciones web para hacer sus servicios aún más accesibles al público.

Como es de esperar, las aplicaciones de karaoke necesitan tener disponible la letra de la canción, así como las marcas de tiempo para que esté sincronizada debidamente. El catálogo de letras más importante del mundo es *Musixmatch,* que da acceso a la letra de millones de canciones verificadas por los sellos discográficos. Dicho servicio no solo se utiliza en aplicaciones de karaoke, sino también en aplicaciones de música como Spotify, para ofrecer servicios similares.

El presente trabajo de fin de grado (TFG) tiene como objetivo desarrollar un servicio web para la conversión de canciones a un formato para karaoke (música con la voz eliminada y letras sincronizadas) utilizando tecnologías basadas en aprendizaje profundo.

En la implementación del servicio, se utilizarán redes neuronales existentes en la bibliografía para realizar la separación voz-música (por ejemplo, *Demucs*), la transcripción/sincronización de las letras (por ejemplo, *Whisper*), la detección del *pitch* o tonalidad de las notas y la corrección de errores de transcripción.

El resultado debe ser un servicio accesible vía web que permita enviar una canción en formato de archivo de audio y que devuelva esa canción en un formato específico para aplicaciones de karaoke.

1.2 Objetivos

Los objetivos principales del presente trabajo fin de grado son los siguientes:

- Desarrollar una aplicación que convierta un archivo de audio introducido a un formato, con una cantidad variable de archivos, que permita su utilización en una aplicación de karaoke.
- La aplicación deberá realizar procesos de separación de voz e instrumental, transcripción y sincronización de las letras, así como corrección de errores de éstos mediante post-procesado.
- Para ello, la aplicación deberá emplear modelos de redes neuronales ya existentes y entrenados, para minimizar el tiempo y recursos invertidos en el entrenamiento de un modelo.
- Además, se ha de desarrollar un servicio web que permita a un usuario enviar a la aplicación el archivo de audio y recibir el resultado del proceso de conversión una vez terminado.

Para lograr la funcionalidad antes expuesta, se han determinado una serie de objetivos secundarios para el desarrollo:

- Análisis de las tareas, determinación de modelos de redes neuronales idóneos para cada tarea, así como el formato de aplicación de karaoke objetivo.
- Estudio de las dependencias o módulos necesarios para realizar las tareas.
- Desarrollo de la aplicación encargada de realizar la conversión.
- Implementación de las funcionalidades requeridas y comprobación del correcto funcionamiento de éstas.
- Adaptación de dichas funciones a una arquitectura cliente-servidor.
- Desarrollo del servicio web que actúa de interfaz de usuario y conexión con la aplicación.
- Despliegue del servicio mediante un servidor estático y comprobación de que se puede acceder a los servicios requeridos.
- Elaboración de la memoria del proyecto.

2 ESTADO DEL ARTE

En los últimos años, la inteligencia artificial ha experimentado un gran crecimiento, desarrollando tecnologías como el *machine learning* gracias, en parte, al desarrollo de algoritmos más sofisticados, como las redes neuronales. Su versatilidad de diseño les da utilidad en muchos campos, como la ingeniería o la medicina.

El lenguaje de programación más popular para el desarrollo, entrenamiento y uso de modelos de inteligencia artificial es Python, gracias a su facilidad de uso y aprendizaje, versatilidad y variedad de librerías que aportan más funcionalidades, entre ellas la generación de entornos virtuales que faciliten la replicación de los entornos en los que se desarrolló la aplicación.

Gracias a esto, se han desarrollado modelos en infinidad de campos. Algunos de los ejemplos más relevantes son los siguientes:

- **Asistentes Virtuales y Chatbots:** Los asistentes virtuales impulsados por IA, como *Siri, Alexa* y *Google Assistant*, han allanado el camino para aplicaciones web que ofrecen interacciones conversacionales más naturales, como GPT 4.0 de OpenAI. Los chatbots basados en IA se utilizan en servicios al cliente, soporte técnico y ventas, proporcionando respuestas rápidas y personalizadas a los usuarios.
- **Reconocimiento de Imágenes y Vídeo:** Los algoritmos de visión por computadora han alcanzado niveles de precisión impresionantes, lo que permite a las aplicaciones web analizar imágenes y videos de manera efectiva. Esto se aplica en campos como la detección en vivo de comportamiento anómalo en videovigilancia, el diagnóstico médico, el reconocimiento facial y la clasificación de contenido multimedia. (1)
- **Recomendación Personalizada**: Los algoritmos de recomendación actuales utilizan técnicas de aprendizaje automático para analizar el comportamiento del usuario y ofrecer sugerencias personalizadas que adecúen la experiencia al usuario y maximicen la retención. Estos algoritmos están presentes en redes sociales (como Facebook o Reddit), plataformas de contenido multimedia (como YouTube o Spotify), portales de noticias, entre otros.
- **Separación de fuentes de audio**: La separación de fuentes de audio es un campo de investigación activo en el que se utilizan técnicas de inteligencia artificial, como redes neuronales convolucionales (CNN) y redes neuronales recurrentes (RNN), para separar y extraer diferentes fuentes de sonido de una mezcla musical. Métodos como *Deep Clustering* y *TasNet* han demostrado resultados prometedores en la separación de voces de música de fondo en grabaciones de audio.

- **Transcripción de audio a texto**: El reconocimiento automático de voz (ASR) ha experimentado avances significativos gracias al aprendizaje profundo y las redes neuronales recurrentes como las redes neuronales de transformadores (BERT). Estas técnicas permiten la transcripción precisa de la voz en texto, lo que resulta fundamental para convertir las letras de las canciones en texto procesable.
- **Detección de pitch y análisis vocal:** La detección de *pitch* es esencial para cuantificar el timbre de las notas que se asocian a la voz del hablante, no solo en un contexto de karaoke. Algoritmos de procesamiento de señales, como el algoritmo de autocorrelación y el algoritmo de *cepstrum*, se utilizan para detectar el pitch de manera eficiente y precisa. Además, el análisis vocal mediante técnicas como la transformada de Fourier y el análisis *cepstral* se emplea para evaluar la calidad vocal y la entonación del hablante, permitiendo así obtener retroalimentación precisa y útil para, por ejemplo, un hablante no nativo.
- **Aprendizaje Automático en el Navegador:** Con el avance de las bibliotecas de aprendizaje automático en JavaScript, como TensorFlow.js y Brain.js, las aplicaciones web pueden ejecutar modelos de IA directamente en el navegador del usuario, haciendo que lleguen a un mayor número de personas. Esto permite experiencias más rápidas y privadas, así como la capacidad de aprovechar los recursos locales del dispositivo si la aplicación lo permite.

En la industria del karaoke, como se ha mencionado anteriormente, la aparición de servicios digitales multiplataforma ha hecho más fácil que nunca establecer un karaoke, tanto para un ambiente casual (una fiesta) como para montar un negocio. Lo que antaño requería una gran cantidad de equipamiento caro ahora se puede conseguir con un dispositivo con conexión a internet y un servicio digital bajo suscripción.

Los servicios bajo suscripción como *Karafun* o *Singa*, están disponibles no solo en aplicaciones de ordenador, sino también en móvil, *Smart TV* e incluso navegadores web. Además, no solo ofrecen un catálogo enorme y diverso de canciones (gracias a catálogos como MusixMatch), sino funcionalidades que lo distinguen de otros servicios similares, como, por ejemplo, una cola de canciones en la que cada usuario puede elegir canciones desde su móvil, descargar canciones para uso offline, cambiar el tono del cantante, etc…

Sin embargo, existen alternativas de código abierto, siendo la más popular *UltraStar*. *UltraStar*, a diferencia de los servicios mencionados anteriormente, es completamente gratuito y permite a sus usuarios añadir nuevas canciones por su cuenta, así como usar canciones que la comunidad crea y comparte. Es un servicio completamente gratuito y personalizable, pero a costa de ser menos conveniente que los servicios de pago y carecer de muchas de las funciones que éstos ofrecen.

El karaoke como mercado tiene unas expectativas de crecimiento bastante favorables, partiendo de una valoración de unos 5.1 mil millones de dólares en 2022 a un crecimiento estable en los próximos años, llegando a valer 6.32 mil millones en 2032 al ritmo de crecimiento que exhibe. (2) Además de los sistemas mencionados anteriormente, son populares los sistemas *all-in-one* con monitor, micrófonos y altavoces inalámbricos y porta-tablets o móviles, que facilitan el despliegue de un karaoke.

El sector más lucrativo en la industria del karaoke es el comercial (70.4% de los beneficios), conformado por restaurantes, pubs y cafeterías que ofrecen karaoke, en especial las noches de karaoke para atraer clientela. El sector doméstico también se encuentra en crecimiento rápido por la creciente adopción del karaoke como medio de entretenimiento en fiestas, lo que incita al desarrollo de servicios más portátiles como los mencionados anteriormente.

Observando los datos de diferentes regiones, la región de Asia-Pacífico fue y sigue siendo una de las más importantes en este mercado, pero Norte América se sitúa en 2022 como la región con la mayor cuota del mercado gracias en parte a la rápida adopción en clubes, restaurantes y fiestas de la región. Europa también es una región con una cuota notable, gracias en gran parte a la adopción en Alemania y Reino Unido. Sin embargo, la región de Asia-Pacífico se considera la región con un mayor crecimiento de entre las observadas.

En conclusión, el karaoke es un mercado lucrativo con unas expectativas de crecimiento bastante notables en múltiples regiones.

3 MATERIALES Y MÉTODOS

En esta sección, se detallan los materiales empleados para el desarrollo de la aplicación y cómo se han utilizado para lograr los objetivos. También se explicará cómo está compuesto el TFG, los componentes desarrollados y su función, así como el servicio REST que se ha implementado.

3.1 Esquema del sistema

A continuación, se mostrará un esquema que ejemplifica el funcionamiento de la aplicación web y su conexión con otros sistemas.

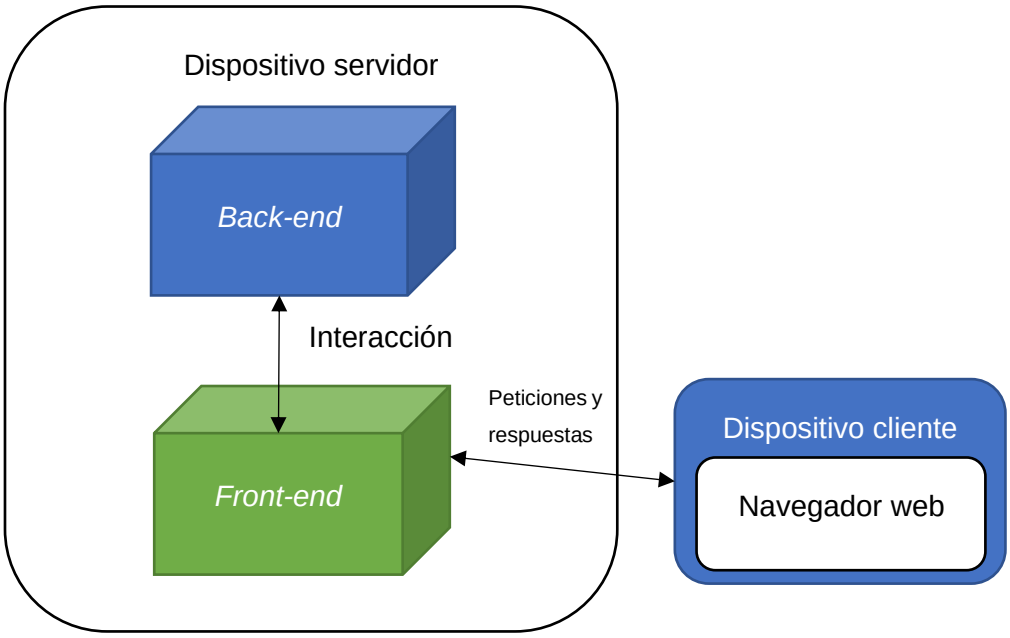

Ilustración 1: Esquema de las interacciones entre sistemas de la aplicación

Al ser una aplicación web, la principal forma que tiene el cliente de interactuar con ella es por medio de un dispositivo con conexión a Internet usando un navegador web. La interfaz de usuario (*front-end*) es desplegada por la respectiva parte de la aplicación, que está integrada dentro del servidor. Utilizando dicha interfaz, el cliente podrá interactuar con la aplicación web y generar peticiones al servidor (*back-end*). Tanto el acceso a la aplicación web como el envío de datos para su conversión se llevan a cabo y se responden mediante peticiones REST.

En los siguientes apartados, se explorarán con más detalle las tecnologías utilizadas para llevar a cabo el desarrollo de la aplicación y se terminará con una explicación más detallada de los procesos que se llevan a cabo en el *back-end* para convertir la información a formato karaoke.

3.2 Lenguajes de programación

En esta categoría, se describen los lenguajes de programación utilizados para desarrollar los elementos que conforman el servicio.

3.2.1 Back-end: Python

Python es un lenguaje orientado a objetos de código abierto que cada vez se está usando más en múltiples industrias tecnológicas, incluyendo el campo de la inteligencia artificial. Prácticamente todos los modelos preentrenados utilizados en este servicio están disponibles como módulos o bibliotecas de Python. Dichas bibliotecas pueden ser instaladas de manera sencilla mediante el gestor de dependencias *pip* e importadas en nuestro código como nos resulte conveniente. Sin embargo, instalar muchos módulos conlleva el riesgo de dañar la arquitectura por defecto de Python y resultaría complicado exportar la gran cantidad de dependencias a diferentes equipos.

Para solucionar este problema, se empleará *conda* para generar entornos virtuales donde instalar todas las dependencias necesarias, con restricciones de versiones, que luego pueden ser exportados a otros equipos, facilitando la portabilidad de la aplicación.

3.2.2 Front-end: JavaScript

JavaScript es el lenguaje de programación orientado a objetos más empleado en el ámbito del desarrollo web. Al igual que Python, podemos introducir dependencias que faciliten la implementación de funciones. Se ha utilizado para desarrollar la interfaz de usuario gracias a React, que tiene compatibilidad con servidores desarrollados en Python.

3.3 Software externo al desarrollo de la aplicación

Esta categoría engloba los programas y plataformas alejados en mayor o menor medida de la aplicación en sí que se han utilizado para facilitar el desarrollo de la aplicación.

3.3.1 Visual Studio Code

Visual Studio Code (3) de Microsoft es el *Integrated Development Environment* (IDE) más utilizado del mundo para desarrollo web. Como cualquier IDE, es un entorno que permite el desarrollo de aplicaciones de manera eficiente. No solo tiene una interfaz intuitiva y personalizable, sino que soporta múltiples lenguajes de programación, así como complementos que facilitan el trabajo al programador y soporte nativo de repositorios en línea. Entre las IDE que también se han considerado se encuentra Pycharm, que es de las mejores IDE para Python, pero se ha optado por Visual Studio para también facilitar el desarrollo de la interfaz de usuario.

3.3.2 GitHub

GitHub (4) es el repositorio web más popular del mundo. En él, se pueden alojar archivos y actualizarlos conforme el desarrollo del proyecto progresa. Así se puede tener

un seguimiento de los cambios que se van haciendo a los archivos que conforman el proyecto y compartirlo con otros usuarios.

3.3.3 Postman

Postman (5) es una plataforma API que permite, entre muchas otras funciones, la prueba del funcionamiento de servicios web en una etapa de desarrollo temprana en la que todavía no se ha desarrollado la aplicación web, permitiendo así detectar errores.

3.3.4 VirtualBox

VirtualBox (6) es un software de virtualización desarrollado por Oracle que nos permite desplegar máquinas virtuales. Dichas máquinas virtuales emulan un ordenador, proporcionando un entorno de trabajo más controlado y separado del resto de la máquina inicial. El único problema es que el hardware de nuestro ordenador es menos accesible (especialmente la GPU) desde el equipo virtualizado, lo que supone un problema a la hora de realizar procesos más dependientes del hardware. Se ha utilizado en fases muy tempranas de desarrollo para comprobar el funcionamiento de los entornos virtuales, pero una vez la aplicación requería acceder a la GPU para realizar procesos, se optó por dejar de utilizarlo

3.3.5 UltraStar

UltraStar (7) es un software de karaoke de código abierto basado en el videojuego *Singstar*. Siendo el programa de código abierto más extendido de este tipo, es el que se ha decidido soportar por medio de la aplicación. Su formato es bastante sencillo de utilizar, ya que está basado en un fichero de texto plano junto con archivos de audio a los que hace referencia y se incluyen en el paquete de archivos devuelto. También cuenta con herramientas de edición que permiten al usuario generar sus propias partituras de karaoke o modificar otras ya existentes. Hay dos versiones que se han tenido en cuenta: *UltraStar Deluxe* y *UltraStar Worldparty*, ya que son las que más difieren en cuanto a formato. Se ha intentado que los ficheros sean compatibles con ambas versiones.

3.4 Back-end (Aplicación de conversión)

El *back-end* se refiere a la parte del servicio que despliega el servicio web, recibe el archivo de audio de la interfaz y realiza el proceso de conversión para convertirlo en un formato compatible con *UltraStar*. Las dependencias utilizadas en el desarrollo son las siguientes:

3.4.1 Anaconda/Mamba

Anaconda (8) es una distribución de Python pensada para facilitar la gestión de paquetes y despliegue de servicios. Los paquetes son manejados por el software *conda*, que nos permite la creación, importación y exportación de entornos virtuales y la gestión

de las dependencias incluidas en dichos entornos, similar a pip, pero minimizando la posibilidad de romper compatibilidad entre dependencias ya instaladas. Mamba (9) es una versión alternativa de conda programada en C++ que aporta un mejor rendimiento. Los instaladores han sido proporcionados por el repositorio de Miniforge (10), que también contiene una interfaz de terminal de comandos (CLI) para ejecutar el script, que da menos problemas que ejecutarlo por la terminal de Visual Studio al poder usar privilegios de administrador para acceder correctamente a los archivos mediante el script.

3.4.2 UltraSinger

UltraSinger (11) es un programa de código abierto actualmente en desarrollo que también busca utilizar modelos de redes neuronales para desarrollar archivos de texto en formato UltraStar. Algunos de sus métodos han resultado de gran utilidad para diseñar el archivo de texto plano en el formato de UltraStar y para definir el traspaso de la información obtenida en los diferentes procesos mediante un formato unificado y simple.

3.4.3 FastAPI

FastAPI (12) es un *framework web* basado en *Starlette* para el desarrollo de APIs con Python. Es muy sencillo de aprender e implementar, rápido comparado con otros *framework* de Python como *Flask* o *requests* y facilita el intercambio de archivos. Estas características nos permiten responder a llamadas de la aplicación web con los resultados del proceso de conversión.

3.4.4 Uvicorn

Uvicorn (13) es una implementación de servidor web ASGI (*Asynchronous Server Gateway Interface*) que permite desplegar un servicio web de manera intuitiva y sencilla, similar a Node.js en JavaScript. Al estar basados en la misma arquitectura, *Uvicorn* se puede utilizar con *FastAPI* para gestionar las peticiones que le lleguen a la aplicación.

3.4.5 Tensorflow

TensorFlow (14) es una biblioteca de código abierto para aprendizaje automático a través de un rango de tareas, y desarrollado por Google para crear sistemas capaces de construir y entrenar redes neuronales para detectar y descifrar patrones y correlaciones, análogos al aprendizaje y razonamiento usados por los humanos. Actualmente es utilizado tanto en la investigación como en los productos de Google frecuentemente reemplazando el rol de su predecesor de código cerrado, *DistBelief*. *TensorFlow* fue originalmente desarrollado por el equipo de Google Brain para uso interno en Google antes de ser publicado bajo la licencia de código abierto Apache 2.0 el 9 de noviembre de 2015.

TensorFlow proporciona una API de Python, así como APIs de C++, Haskell, Java, Go y Rust. También hay bibliotecas de terceros para C#, Julia, R, Scala y OCaml (14)

Estas cualidades son muy útiles, pero debido a la cantidad y complejidad de los modelos que estamos utilizando, así como de los recursos que utilizan, se ha optado por una inferencia del lado del servidor. Sin embargo, sigue siendo utilizado para controlar el uso de recursos de nuestro servidor.

3.4.6 ImageMagick:

ImageMagick (15) es un programa de código abierto que contiene múltiples herramientas para la manipulación de imágenes de manera compleja y automática mediante scripts. Está disponible como software o dependencias de Python. Estas herramientas son usadas por moviepy para generar el fondo del video si el usuario elige esa opción.

3.4.7 Prosodic:

Prosodic (16) es un analizador métrico-fonológico escrito en Python. Esta clase de programas analizan textos y los separan por estrofas, versos, palabras, sílabas y fonemas. Para ello, suelen utilizar diccionarios, por lo que su funcionamiento está limitado a los idiomas para los que están diseñados para dar soporte. En el caso de Prosodic, contiene diccionarios para inglés y finlandés, pero se pueden añadir más idiomas introduciendo sus diccionarios en el formato correcto. Se puede implementar además como una dependencia de Python, facilitando su implementación en servicios de back-end. Al ser un analizador basado en un diccionario, su funcionamiento es enteramente determinista, es decir, el mismo texto tendrá siempre los mismos resultados. Para este proyecto, nos interesa utilizar estos programas para obtener las sílabas de la transcripción de Whisper y realizar las notas a nivel de sílaba, asociando las palabras a los eventos de notas registrados por basic_pitch. Esto nos permite obtener notas a nivel de sílaba, cosa imposible solo mediante Whisper, a riesgo de comprometer la precisión de la transcripción resultante, ya que inevitablemente van a haber imprecisiones al asociar sílabas a las notas.

3.4.8 Pyverse:

Pyverse (17) es un algoritmo silabeador de textos en español escrito en Python. Al igual que Prosodic, utiliza un diccionario interno para definir las sílabas, por lo que sus resultados son deterministas. Este algoritmo tiene en cuenta además las normas de la poesía a la hora de contar sílabas, es decir, la sinalefa y los finales de verso. Sin embargo, se ha optado por obtener las sílabas sin seguir estas convenciones, ya que el proceso de asignación de sílabas a notas está vinculado a las palabras. Se puede utilizar como una dependencia e implementar fácilmente en scripts.

3.4.9 Librosa:

Librosa (18) es una dependencia de Python para el análisis de archivos de audio. Nos permite realizar toda clase de procesado de señal, pero nos interesa usarlo para detectar el tempo global (en BPM) del audio para introducirlo en el fichero final.

3.4.10 Moviepy:

Moviepy (19) es una dependencia de Python centrada en la creación, edición, composición y procesado de video, desde las funciones más básicas hasta la creación de efectos avanzados. Es compatible con múltiples formatos de video. En esta práctica, se utilizará para crear vídeos *karaoke*, es decir, con instrumental y la letra en pantalla. También abre la puerta a una compatibilidad con archivos de vídeo en el futuro.

3.4.11 Modelos preentrenados de inteligencia artificial

Los procesos integrales para el adecuado procesado de audio se han llevado a cabo por medio de modelos preentrenados de algoritmos de inteligencia artificial. Los métodos que permiten la descarga y uso de dichos modelos se pueden instalar por medio de sus respectivas dependencias de Python, haciéndolos muy fáciles de instalar e implementar. El hecho de que sean modelos preentrenados para su respectiva funcionalidad mediante un *dataset* considerable nos ahorra los extensos procesos relacionados con dicho entrenamiento (recopilación del *dataset*, elección del método de entrenamiento, monitorizado del aprendizaje, etc…) y nos permite utilizarlos directamente. Además, algunos algoritmos cuentan con modelos de diferentes requisitos técnicos entre los que elegir en función de nuestra situación.

3.4.11.1 Whisper

Whisper (20) es un modelo de reconocimiento de habla desarrollado por OpenAI. Está entrenado por un dataset enorme y diverso con una gran cantidad de idiomas. Además de realizar transcripciones monolingües, también se ha entrenado para realizar transcripciones multilingües, traducción al inglés e identificación de idioma. A diferencia de la versión por API de Whisper, la versión disponible en el repositorio de Github de OpenAI se puede instalar y usar de manera gratuita como una dependencia y cuenta con cinco modelos multilingües entrenados y listos para usarse e implementarse en scripts de Python.

Respecto a lo precisa que es su transcripción, Whisper es uno de los modelos de transcripción más precisos actualmente. La métrica más utilizada para calcular la precisión de una transcripción es la tasa de error de palabra o WER, que consiste en la relación entre el número de errores (sustituciones, eliminaciones e inserciones) relativo al número total de palabras:

$$WER = \frac{Sustituciones + Eliminaciones + Inserciones}{Nºpalabras}$$

Las pruebas realizadas en el Open ASR Leaderboard (21) reportan un WER medio para transcripciones en inglés de 8.06% para la versión 2 y de 7,7% para la versión 3, lo que lo sitúa como el segundo modelo más preciso del mundo, solo situado por debajo de algunos modelos de Nvidia Canary, que fueron lanzados en abril de 2024. Como referencia, este desempeño lo sitúa por encima de muchos otros modelos ASR y hasta de transcriptores humanos. (20)

En lo que se refiere a compatibilidad con múltiples idiomas, el modelo está entrenado en 99 idiomas, pero principalmente en inglés. Es importante indicar que el desempeño del programa varía notablemente en función del idioma utilizado. En la siguiente gráfica, se indican los resultados de WER en diferentes idiomas y datasets. El desempeño de Whisper es excelente en idiomas extendidos como el español, italiano e inglés, pero deja bastante que desear en idiomas mucho menos extendidos o de los que apenas se tiene información de entrenamiento.

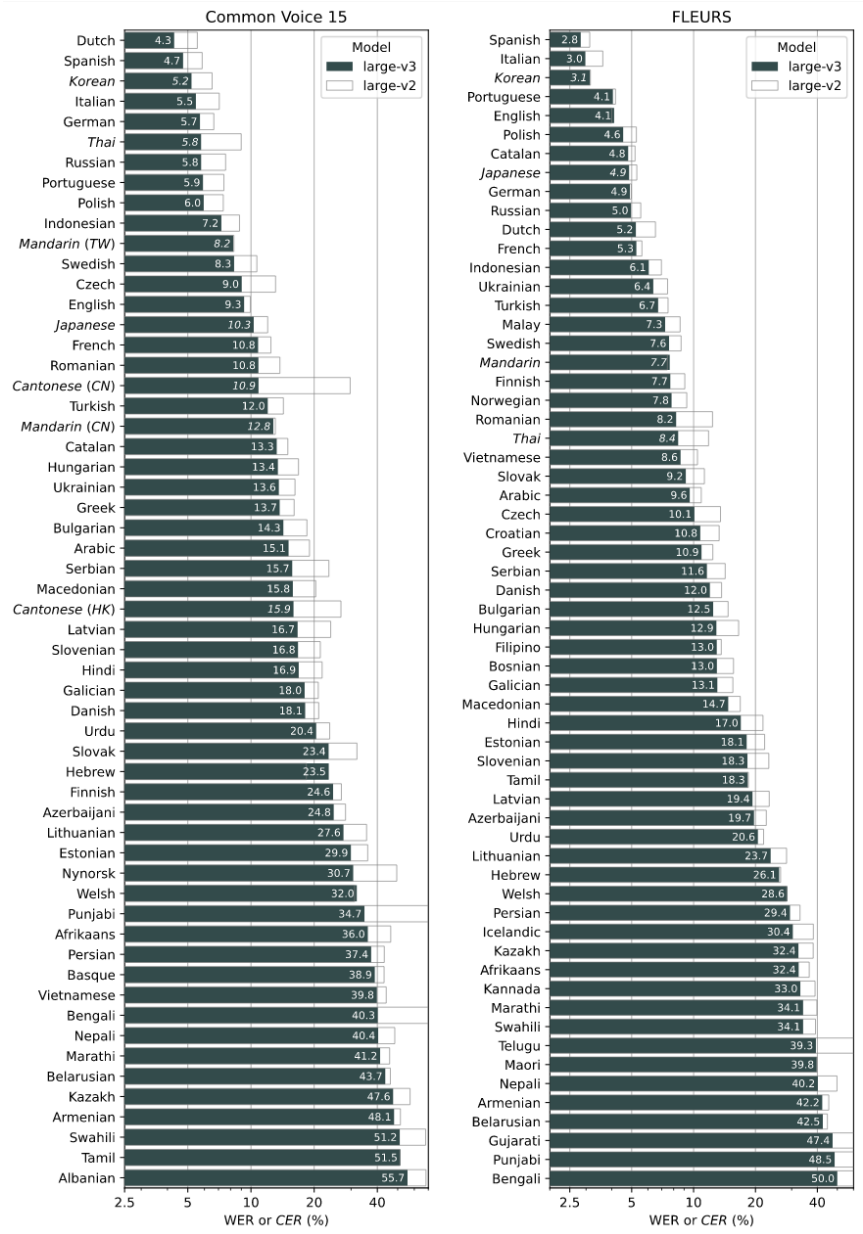

Ilustración 2: Comparativa de WER de transcripciones de Whisper en función del idioma, modelo y dataset

Cuanto más grande es el modelo, más preciso es en su transcripción, pero requiere más recursos (VRAM) y es más lento. De esta manera, podemos elegir el modelo que mejor se ajuste a nuestra aplicación. Para hacer más fácil elegir un modelo grande sin sacrificar demasiados recursos, se ha optado por utilizar WhisperX (22).

WhisperX es un *fork* de Whisper que permite acceder a los mismos modelos con notables mejoras en cuanto a uso de memoria. Esto se debe al uso de *faster-whisper* (23) como *backend*, una reimplementación del modelo de Whipser en CTranslate2, una interfaz para modelos mucho más eficiente. La siguiente tabla es una comparativa de la velocidad de modelos *large-v2* (uno de los modelos más potentes de Whisper) para una transcripción de 13 minutos, así como su uso de recursos. Como podemos observar, *faster-whisper* no

solo es mucho más rápido, sino que utiliza menos recursos para el mismo modelo. Tanto es así que, mientras que con el Whisper normal solo podríamos usar el modelo *medium*, con WhisperX es posible utilizar el modelo *large-v2* con el mismo equipo:

Implementación	**Precisión**	***Beam size***	**Tiempo**	**Memoria GPU máxima usada**	**Memoria CPU máxima usada**
openai/whisper	fp16	5	4m30s	11325MB	9439MB
faster-whisper	fp16	5	54s	4755MB	3244MB
faster-whisper	int8	5	59s	3091MB	3117MB

Tabla 1: Tabla comparativa de implementaciones de modelos large-v2 sobre GPU

Otra mejora proporcionada por WhisperX es la capacidad de implementar alineamiento forzado por medio de modelos ASR a nivel de fonema como wav2vec2.0. Esto soluciona un problema de Whisper, que es el uso marcas de tiempo a nivel de habla y no de palabra, lo que puede resultar en imprecisiones del orden de segundos. Originalmente, se pensó utilizar crepe, Timething o Montreal Forced Aligner para el alineamiento forzado, pero crepe solo era compatible con archivos .wav y Timething y MFA no aportaban los resultados esperados.

WhisperX también está mejor equipado para hacer frente a alucinaciones, es decir, transcripciones “fantasma” cuando el vídeo tiene silencio o partes sin voz, especialmente al final (Por ejemplo, un *¡Suscríbete al canal!* o una marca de agua de audio cuando no están en el audio original). Para ello, segmenta el audio por medio de un filtro VAD o *Voice Activity Detection*. Este filtro se usa para detectar en qué regiones del audio introducido hay habla, y a partir de esas regiones realizar la transcripción, minimizando así efectos de contorno y mejorando la eficiencia del sistema al permitir transcripciones en paralelo. Además, las operaciones de cortado y mezclado de segmentos muy cortos que realiza el VAD hacen que el modelo evite bucles de transcripción y alucinaciones durante regiones de habla inactivas. Sin embargo, si los valores de umbral del VAD son demasiado estrictos, puede haber segmentos de habla no detectados, mientras que, por otro lado, unos valores demasiado laxos no solucionan las alucinaciones.

Para mostrar la precisión de esta implementación, en la siguiente tabla se muestra el desempeño de los modelos wav2vec2.0, el modelo large-v2 de *Whisper* y *WhisperX* en un escenario de transcripción a nivel de palabra de audio de larga duración. *Spd.* se refiere a la velocidad de transcripción relativa, WER se refiere a la tasa de error de palabra, *IER* se refiere a la tasa de error de inserción y 5-Dup se refiere a la cantidad de duplicados.

Model	TED-LIUM [25]				Kincaid46 [26]			AMI [23]		SWB [24]	
	Spd.↑	WER↓	IER↓	5-Dup.↓	WER↓	IER↓	5-Dup.↓	Prec.↑	Rec.↑	Prec.↑	Rec.↑
wav2vec2.0 [2]	10.3×	19.8	8.5	**129**	28.0	5.3	**29**	81.8	45.5	92.9	54.3
Whisper [9]	1.0×	10.5	7.7	221	12.5	3.2	131	78.9	52.1	85.4	62.8
WhisperX	**11.8**×	**9.7**	**6.7**	189	**11.8**	**2.2**	75	**84.1**	**60.3**	**93.2**	**65.4**

Ilustración 3: Comparativa de modelos de transcripción bajo diferentes corpus lingüísticos

Este modelo se utilizará para obtener una transcripción del archivo de audio proporcionado por el usuario, a partir de la cual se obtiene la letra y las marcas de tiempo necesarias para sincronizarla con la canción. Después, se aplicará alineamiento forzado para alinear la transcripción con los fonemas y tener una transcripción más precisa y marcas a nivel de palabra. La siguiente imagen muestra un diagrama de los procesos descritos anteriormente:

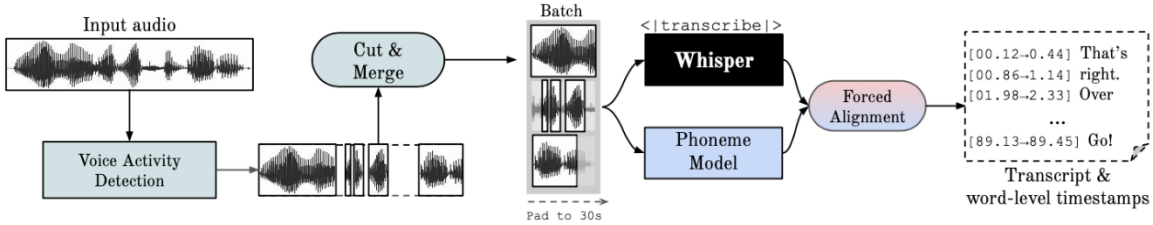

Ilustración 4: Diagrama del funcionamiento de WhisperX

Al igual que la transcripción de Whisper, la calidad del alineamiento forzado de WhisperX depende del idioma y es mucho más limitado, solo teniendo alineamiento para inglés, francés, alemán, español e italiano. En el caso del modelo wav2vec2.0, comprobaremos su precisión mediante múltiples parámetros. Primero, comprobaremos el % de tasa de error de palabra (WER) de varios datasets, versiones y disposiciones del modelo. Como se puede observar, los modelos más entrenados y con datasets limpios tienen un desempeño comparable a Whisper. (24)

Dataset	Model	No LM	full LM		quantized LM		Server with LM
			b_w 10	b_w 100	b_w 10	b_w 100	
test-clean	100hr quantized	6.8	4.8	4.6	5.1	4.8	
	100hr	6.4	4.9	4.6	4.8	4.6	3.4
	960hr quantized	4.0	3.3	3.3	3.3	3.4	
	960hr	3.7	3.1	3.1	3.1	3.1	2.6
test-other	100hr quantized	15.5	11.2	10.4	11.5	10.4	
	100hr	14.9	11.2	11.0	11.2	11.0	8.0
	960hr quantized	10.0	7.5	7.2	7.5	7.2	
	960hr	9.0	7.6	7.4	7.6	7.4	6.1
dev-clean	100hr quantized	7.8	4.7	4.3	4.7	4.3	
	100hr	7.0	4.4	4.1	4.4	4.1	2.7
	960hr quantized	4.7	3.3	3.2	3.2	3.1	
	960hr	4.2	3.2	3.2	3.2	3.2	2.0
dev-other	100hr quantized	15.8	11.0	9.9	11.0	9.8	
	100hr	15.0	10.7	9.8	10.6	9.7	7.9
	960hr quantized	11.0	8.7	8.1	8.7	8.2	
	960hr	10.3	7.8	7.5	7.8	7.5	5.9

Tabla 2: Porcentaje de tasa de error de palabra (WER) para diferentes modelos

Además, también se ha comprobado su precisión a la hora de detectar fonemas en el contexto de un programa para detectar errores de pronunciación. (25) Para ello, se emplearán dos estadísticos, el valor F1 y ACC para cada palabra estudiada. El valor F1 se calcula en base a la precisión (PRE) y exhaustividad (REC) de cada modelo para la palabra indicada en el eje x, con las siguientes expresiones:

$$REC = \frac{TP}{TP + FN}$$

$$PRE = \frac{TP}{TP + FP}$$

$$F1 = 2 * \frac{PRE * REC}{PRE + REC}$$

Donde TP, FP y FN son los verdaderos positivos, falsos positivos y falsos negativos respectivamente. El valor ACC (%) es simplemente el inverso de la tasa de error del programa. La siguiente gráfica compara el funcionamiento de múltiples modelos pre-entrenados de Wav2vec en esta tarea:

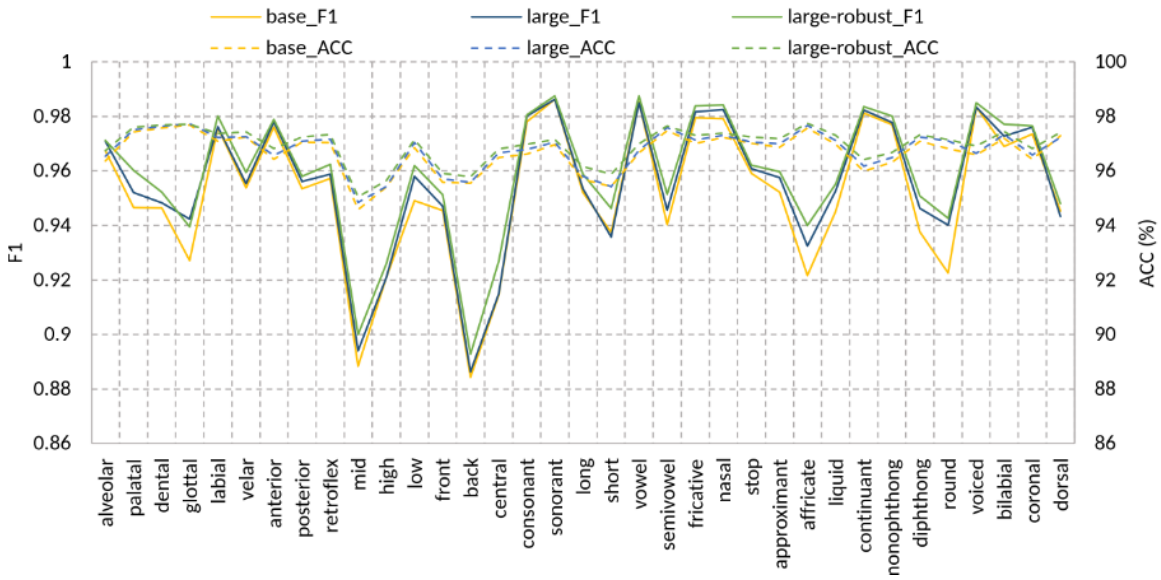

Ilustración 5: Desempeño de tres modelos pre-entrenados en una tarea de reconocimiento de habla

3.4.11.2 Demucs:

Demucs (26) es un modelo de separación de fuentes originalmente desarrollado por Meta, pero actualmente está bajo Kyutai. Este modelo permite separar la percusión, los bajos y la voz del resto del acompañamiento musical de una pista de audio y obtenerlos en diferentes archivos.

En su repositorio, se pueden encontrar muchos modelos que han sido entrenados por medio del dataset MusDB HQ, además de 800 canciones adicionales. Dichos modelos se pueden incorporar con facilidad a scripts de Python e implementar como si de una función se tratara, incluyendo la opción de solo separar voz e instrumental.

En cuanto a precisión de separación, Demucs se sitúa favorablemente en comparación con otros modelos. Las métricas utilizadas para medirla son la relación señal a distorsión (SDR) media de las 4 salidas, el Mean Opinion Score (MOS) respecto a la calidad de la separación y ausencia de artefactos percibida por oyentes humanos y el MOS respecto a la contaminación por otras fuentes. En estas pruebas, demucs obtuvo un SDR de 7 a 9 dB en función de la versión y una valoración favorable. (26)

Estas características son la razón por las que este modelo es uno de los más utilizados en su campo y por tanto se ha elegido para obtener la parte instrumental para el karaoke.

Sin embargo, uno de los problemas más comunes de las implementaciones de Demucs originales era que, incluso cuando realizabas una separación vocal-instrumental, el sistema realizaba 4 separaciones una a una y combinaba para obtener 2 salidas. En las implementaciones de *Hybrid Demucs*, el programa no solo puede elegir entre el dominio de forma de onda o espectrograma para realizar la separación, sino que también puede

realizar las separaciones al mismo tiempo, lo que minimiza considerablemente el tiempo de ejecución. Una separación que llevaría 10 minutos en el modelo *mdx_extra* se realiza en unos 5 minutos en el modelo *htdemucs_mmi*. La siguiente imagen muestra un diagrama de la arquitectura de *Hybrid Transformer Demucs*, un modelo más avanzado que implementa un *Transformer Encoder* que trabaja en ambos dominios.

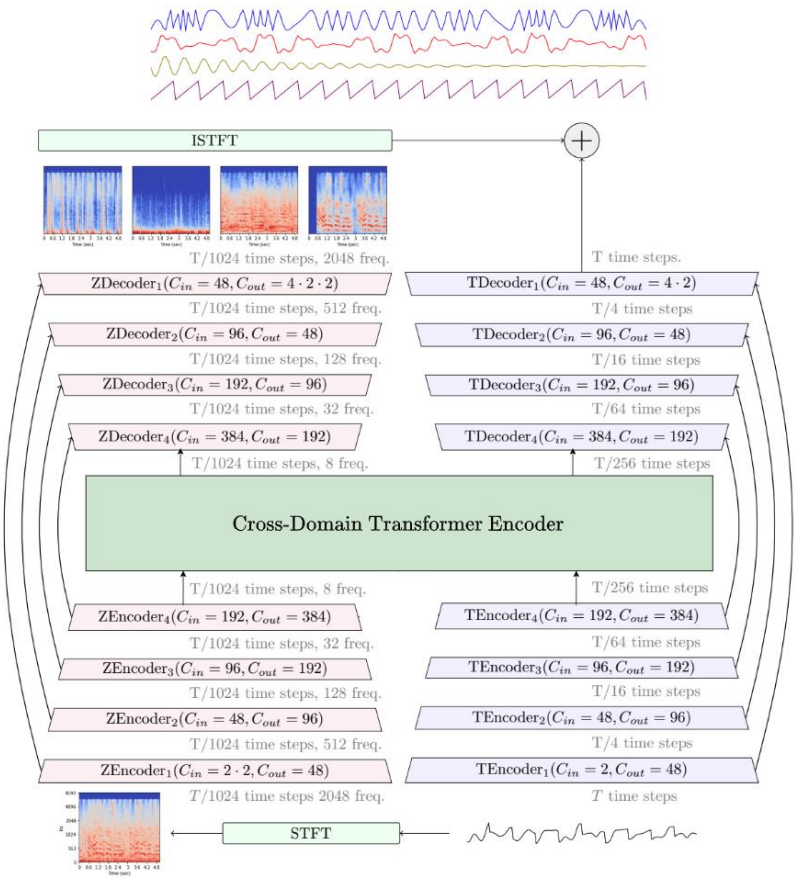

Ilustración 6: Esquema de la arquitectura del modelo Hybrid Transformer Demucs

El modelo Hybrid Transformer Demucs se compone de dos redes U independientes, el mismo tipo de redes neuronales que se usan en segmentación de imágenes en biomedicina; una realiza convoluciones en el dominio del tiempo (forma de onda), y la otra realiza convoluciones en el dominio de la frecuencia (espectrograma). Cada red U se compone de 4 capas de codificado y 4 de decodificado con diferentes coeficientes. Entre ellas se encuentra un *Cross-Domain Transformer Encoder*, un componente multi-capa que recoge la información de ambos dominios en paralelo y alterna entre un *Cross-Attention Encoding* sobre ambas señales, proceso de codificado que tiene en cuenta ambas señales para la salida en su respectivo dominio, y un *Transformer Encoder*. Al final de estos procesos, la salida de la red en el dominio de la frecuencia se pasa al dominio del tiempo mediante una Transformación de Fourier de Tiempo Corto inversa (iSTFT) y se combina con la salida en el dominio del tiempo, dando lugar a la predicción del modelo.

3.4.11.3 Basic Pitch:

Basic Pitch (27) es un conversor de audio a MIDI de código abierto desarrollado por Spotify. Puede funcionar con cualquier instrumento (aunque a nosotros nos interesa trabajar solo con la voz, que es monofónica), exportarse a diferentes formatos, incluyendo un CSV con los pitches de las notas y detectar *pitch bend* (variación del tono), el cual, si bien aporta información, es difícil incorporarlo a la escritura del formato UltraStar debido a su número de columnas irregular.

El modelo funciona de manera superior a otros modelos de detección de pitch de instrumentos como MI-AMT tanto para fuentes polifónicas como monofónicas. En la siguiente tabla, se compara la precisión de nota a nivel de cuadro (Acc), la medida F a nivel de nota (F) y la medida F a nivel de nota ignorando el offset (Fno), siendo el último el estadístico más importante para definir la precisión del modelo, de diferentes versiones del modelo NMP (basic-pitch) y MI-AMT. (28)

	Molina			**GuitarSet**			**Maestro**			**Slakh**			**Phenicx**		
	Acc	Fno	F	Acc	Fno	F	Acc	Fno	F	Acc	Fno	F	Acc	Fno	F
MI-AMT	.48	.31	.11	.43	.59	.27	.39	.30	.07	.40	.23	.07	.13	.12	.05
NMP	**.63**	.52	.35	**.70**	**.79**	**.56**	.38	.71	.11	**.44**	.42	.21	**.53**	.49	.35
NMP - P	.60	.55	.38	.67	.78	.55	.36	.65	.12	.40	**.43**	**.23**	.50	**.51**	**.36**
NMP - H	.45	.36	.20	.50	.65	.40	.27	.48	.10	.33	.36	.17	.37	.39	.23

Ilustración 7: Comparativa de precisión de modelos de detección de pitch

Este es el modelo que usaremos para detectar el tono de las notas a lo largo de la canción, el cual es indispensable para generar el fichero final.

3.5 Interfaz de usuario

La interfaz de usuario o *front-end* es la parte más gráfica de la aplicación. Su función consiste en proveer de una interfaz que permita a los usuarios introducir archivos de audio y obtener los ficheros necesarios para una aplicación UltraStar.

3.5.1 React

React (29) es una biblioteca para aplicaciones cliente basada en JavaScript utilizada para diseñar interfaces de usuario interactivas. Está basada en componentes con estados que se pueden incluir dentro de otros para generar vistas de manera modular. Los componentes resultantes se pueden expresar con JSX, una sintaxis de JavaScript muy similar a XML, mezclada con elementos HTML, lo que hace su diseño e implementación bastante intuitivo.

Estas cualidades hacen que React sea una de las bibliotecas más populares para el diseño de páginas web incluso hoy en día, siendo utilizada por multitud de empresas y desarrolladores en el mundo (30).

3.5.1.1 ¿Cómo se crea una aplicación web de React?

Al principio se puede pensar que construir la aplicación web para poder conectarse y tener un entorno de desarrollo requiere mucho trabajo. Pero la realidad es que, con un solo comando y mínima configuración previa, podemos empezar con una base muy sólida y centrarnos en diseñar los componentes y enrutarlos. Dicho comando es '*npx create-react-app <nombre de la app>*'.

Esto nos genera un fichero con lo necesario para que, una vez ejecutemos el comando '*npm start*' en esa carpeta, nos inicie la aplicación web en el puerto 3000 en un entorno de desarrollo, donde mientras esa aplicación se esté ejecutando, podremos ver los cambios que hagamos en los archivos sin tener que apagar la aplicación. Además, no tenemos que estar completamente aislados del servidor. Se puede diseñar la aplicación para que se esté conectando con la aplicación servidor y así comprobamos que la aplicación funcione. Esto requiere que configuremos el CORS (*Cross-Origin Resource Sharing*) en el servidor para que la política del navegador SOP (*Same Origin Policy*) no bloquee la conexión entre aplicación cliente y servidor al ser de diferentes dominios.

Una vez completada la aplicación, ejecutaremos el comando '*npm run build*'. Este comando genera en una nueva carpeta llamada *'build'* una serie de archivos que constituyen la aplicación web en una versión de lanzamiento o *'release'*. Copiamos esta carpeta en el directorio donde tenemos la aplicación y definimos dentro de nuestro script esa carpeta como aquella donde pondremos una instancia de recursos estáticos de *FastAPI* llamada *StaticFiles* (con cuidado de que esté después de los procesos de conversión porque puede afectar su funcionamiento). De esta forma, cuando despleguemos nuestro servidor, éste utilizará como interfaz la aplicación cliente que hemos creado con React y los usuarios podrán usarla conectándose directamente al servidor, haciendo mucho más sencillo desplegar la aplicación en su totalidad.

3.5.1.2 Material UI

De entre las librerías a las que da acceso React, una de las que más se han usado en el desarrollo de la interfaz es *Material UI* (31). Consisten en una serie de componentes preconstruidos aplicando los principios del estándar *Material Design* de Google. Esto nos permite crear una página web que se asemeje lo máximo posible en diseño y manejo a una página web convencional. También nos ahorra tener que definir exhaustivamente la mayor parte de la funcionalidad de los componentes ya que viene integrada en los mismos y es fácilmente modificable para adaptarla al comportamiento esperado del componente o al diseño que queremos buscar en la página. Además, como hemos dicho antes, React es de

naturaleza modular, por lo que implementar estos componentes una vez importada la dependencia es simple. También se incluyen plantillas de diseños de plataformas web.

3.5.2 Fetch

Fetch es una API para JavaScript que permite enviar peticiones asíncronas a un servidor. En esta aplicación, la utilizamos para permitir al usuario enviar el archivo de audio a la aplicación de conversión y recibir, procesar y descargar el resultado de la conversión.

3.6 Explicación del TFG y sus componentes

En esta categoría se explicará cómo está hecho el TFG y los componentes implementados durante su desarrollo. También se explicarán los servicios REST que se han implementado en la aplicación.

3.6.1 Back-end

El *back-end* es la parte de la aplicación web que se ejecuta en el lado del servidor. Se despliega por medio de FastAPI en conjunto con Uvicorn y ejecuta las siguientes funciones:

- Conectar con la interfaz de usuario del *front-end* y recibir los archivos de audio provistos por el usuario desde ésta.
- Realizar todo el procesado y post-procesado necesario para generar el paquete de UltraStar o el vídeo requerido.
- Enviar el resultado de vuelta a la interfaz de usuario.
- Mantener la conexión activa al recibir solicitudes.

Esta parte de la aplicación es la que ha sido desarrollada primero. Los archivos y directorios que lo conforman se pueden observar en la siguiente imagen:

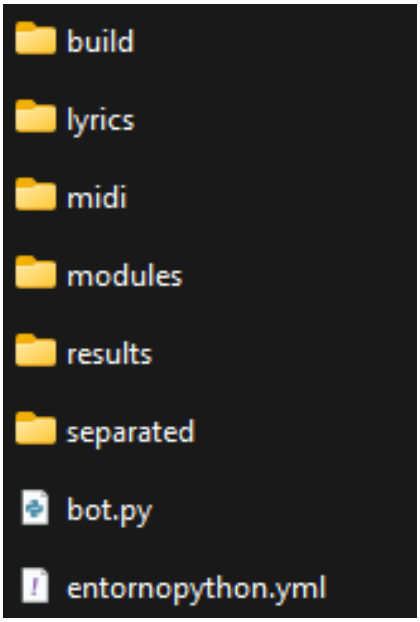

Ilustración 8: Carpeta donde se almacena la aplicación web

A continuación, se detalla la función de cada directorio y archivo:

- **build:** Carpeta donde se encuentran los recursos que constituyen el *front-end*.
- **lyrics:** Carpeta donde se guarda la letra en formato JSON durante el proceso de generación de video.
- **midi:** Carpeta donde se guarda temporalmente el archivo .csv con los resultados del cálculo de *pitch* de *Basic Pitch*.
- **modules:** Carpeta que incluye las funciones externas al archivo principal que utiliza el programa.
- **results:** Carpeta donde se almacenan los archivos resultantes del proceso de conversión.
- **separated:** Carpeta generada automáticamente por *Demucs* donde se almacenarán los archivos resultantes de la separación de voz e instrumental.
- **bot.py:** Script principal del proyecto, donde se ejecutará el despliegue de la aplicación web.
- **entornopython.yml:** Archivo YAML que contiene las dependencias del entorno virtual en el que se desarrolló y probó la aplicación para facilitar su despliegue en otro equipo.

3.6.2 Interfaz de usuario

La aplicación cliente o *front-end* tiene como función proporcionar una interfaz clara e intuitiva a los usuarios que les permita interactuar con la aplicación y acceder a los servicios de ésta. Para su desarrollo, se ha generado una aplicación React con entorno de desarrollo separado del servidor mediante los procesos indicados anteriormente. La aplicación se compone de una única vista, ya que se ha considerado que una estructura de una sola página es suficiente para aportar el servicio propuesto.

3.6.2.1 Vista

URL relativa: /

Archivo local: /src/webclient/src/App.js

Esta vista es la única vista de la aplicación, haciendo que la aplicación tenga solo una página. Además de información para el usuario, la página se compone de un selector que permite al usuario decidir si quiere recibir un archivo de UltraStar o un video para karaoke y un botón que permite al usuario enviar el archivo de audio al *back-end*. Sin salirse de la página, el servidor le devolverá el tipo de archivo solicitado a partir de su introducción.

3.6.3 Servicio REST

Los servicios REST hacen de intermediarios en una arquitectura cliente-servidor, como la que se forma entre nuestro *front-end* (interfaz de usuario) y *back-end* (aplicación de conversión). Por tanto, son especialmente útiles para el traspaso de datos utilizando el

formato JSON. A continuación, se explican las funcionalidades que se ofrecen por medio de servicios REST con el siguiente formato:

Servicio (Método y URL):

Datos que enviar (si no se especifica dónde, se asume que es en el cuerpo de la solicitud).

Funcionamiento.

Datos devueltos (en caso de que se haga con éxito).

- **Servicio POST /video:**
  - Datos que enviar:

    {
    “inputfile”: File,
    “video_output”: String
    }
  - Funcionamiento: El valor *video_output* se lee para determinar el proceso para llevar a cabo a modo de booleano. FastAPI permite recibir un String que se procesa como un booleano que es True si se recibe “yes”, “on”, “1”, “true”, sin diferenciar mayúsculas. En cualquier otro caso el valor es False. Si el booleano es True, se ejecuta el proceso para enviar un video mp4 con instrumental y letra en pantalla, que consiste en:
    - Obtención de transcripción a nivel de verso
    - Separación de instrumental y voz
    - Generación del vídeo
    - Envío del vídeo resultante

    Si por el contrario se recibe False, se ejecuta el proceso para devolver un paquete para UltraStar, que es:
    - Separación de instrumental y voz
    - Obtención de transcripción a nivel de palabra
    - Obtención de *pitch* de las notas
    - Separación de la letra en sílabas
    - Asociación de las notas a sílabas
    - Generación del archivo de texto en formato UltraStar
    - Compresión del archivo de texto y los ficheros de audio relacionados
    - Envío del archivo comprimido.

    Dependiendo del hardware del servidor, se estima que el proceso completo dura entre 20 y 30 minutos. En ambos casos, se utiliza el archivo de audio enviado en el atributo “inputfile”.

- Datos devueltos: Un vídeo mp4 o un archivo comprimido en función del valor de *video_output*.

3.7 Flujo de la aplicación

Para finalizar esta sección, en este apartado se explicarán con más detalle los diferentes procesos que tienen lugar en la conversión de archivos, así como las librerías que se emplean en cada parte. El siguiente diagrama de flujo indica el funcionamiento de la aplicación desde que se envía el POST /video con el formato explicado en el apartado anterior hasta que se envía el resultado de la conversión y se borran la mayoría de los archivos guardados durante el proceso:

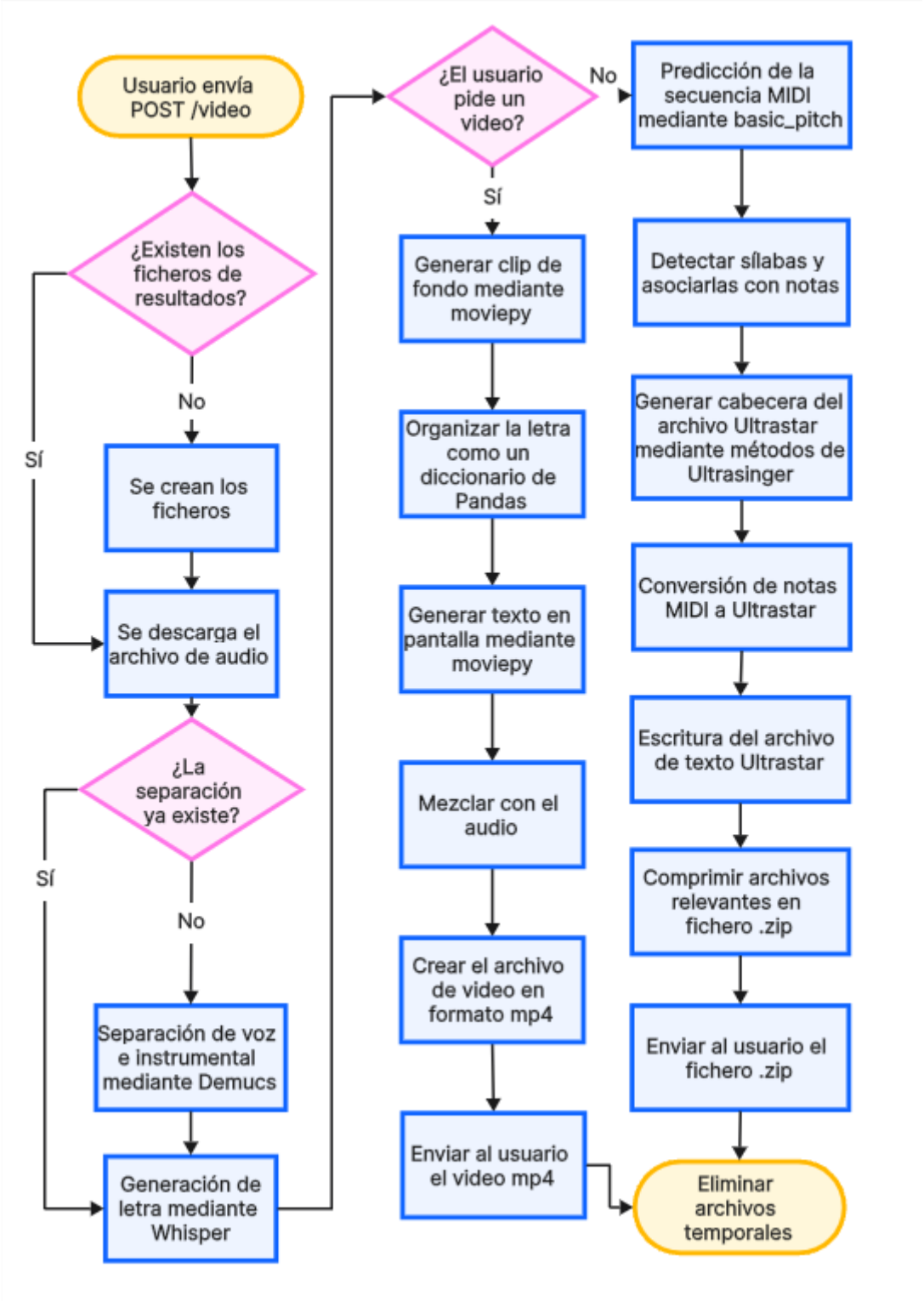

Ilustración 9: Diagrama de flujo del funcionamiento de la aplicación

3.7.1 Recepción del POST:

Una vez el *back-end* recibe el POST del usuario, se comprueba si, en el directorio de trabajo, aparecen las carpetas donde se guardan resultados de procesos, es decir, *results*, *lyrics* y *midi* (*separated* no es necesario porque Demucs la crea si no existe) y se crean en caso de que no existan. Después, se descarga el archivo de audio de la petición y se guarda en *results*. Después, se inicializa la clase en la que se definen los métodos utilizados.

3.7.2 Separación de voz e instrumental:

El siguiente paso es separar las fuentes pertenecientes a la voz y al instrumental del archivo de audio. Este proceso se realiza mediante el modelo pre-construido *htdemucs_mmi* de Demucs. Cabe mencionar que, debido a que este es el proceso más largo del programa con diferencia, se ha optado por implementar la posibilidad de saltarlo si ya se tiene la separación hecha de un archivo de audio del mismo nombre. Después de este proceso, se realizarán diferentes procedimientos según si el usuario pide un archivo UltraStar o un vídeo.

3.7.3 Generación de letra:

Una vez se tienen los archivos y carpetas, se empieza generando la letra. Este proceso se realiza mediante el modelo pre-construido *large-v3* de Whisper cargado mediante WhisperX para mejor eficiencia. Este modelo recorrerá el archivo de audio, inferirá el idioma automáticamente y realizará una transcripción. Dependiendo de si el usuario solicita un video o un paquete UltraStar, la transcripción se hará de forma ligeramente diferente:

- Si nos pide un vídeo, se guardará como un JSON con marcas de tiempo a nivel de frases de tamaño algo más reducido (para que encaje en el video).
- Si nos pide un paquete UltraStar, realizamos la transcripción con *chunks* del tamaño por defecto y realizamos alineamiento forzado mediante WhisperX, obteniendo además el idioma detectado de la canción.

3.7.4 Proceso de cálculo de pitch:

Este proceso y el siguiente se usan solo cuando el usuario pide un archivo UltraStar (es irrelevante para el vídeo). Este proceso predice la secuencia MIDI equivalente a la voz y el *pitch* de cada nota, utilizando el modelo *basic-pitch*. Esta secuencia se puede usar más adelante para calcular el *pitch* de las notas de UltraStar, ya que la relación entre ambas es lineal. Como referencia, la nota C4 (Do) en formato MIDI es 48, mientras que en UltraStar es 0.

3.7.5 Proceso de obtención de sílabas:

Para separar las palabras de la transcripción en sílabas, utilizamos Pyverse o Prosodic según si el texto está en español o inglés respectivamente. En ambos casos usamos la transcripción sin marcas de tiempo y analizamos para obtener una lista en la que cada palabra esté desglosada en sílabas.

3.7.6 Proceso de asociación de sílabas a notas:

Este proceso iterativo intenta asociar las sílabas a las notas obtenidas mediante *basic-pitch*. En cada iteración, se busca si hay notas que corresponden al intervalo de la palabra marcado por WhisperX basado en el comienzo de éstas:

- Si hay el mismo número de notas que de sílabas, se asocian directamente las notas a las palabras.
- Si hay más sílabas que notas, las sílabas que no tienen nota asociable se añaden a la última nota.
- Si hay más notas que sílabas, las notas sin sílaba asociada se les añade un apóstrofo (~), que el Whisper interpreta como una continuación de la sílaba anterior.
- Si no hay notas en el intervalo de esa palabra (muy posible en palabras muy cortas), añadimos tantos eventos de nota como sílabas tenga la palabra repartidos en el intervalo de la palabra. Para el pitch utilizamos la nota más próxima.

Una vez obtenidas las notas, pasamos los resultados al formato *TranscribedData* para generar la canción de UltraStar.

3.7.7 Proceso de generación del TXT de UltraStar:

Una vez obtenido el audio instrumental y vocal, la transcripción y el *pitch*, podemos crear el archivo .txt que es imprescindible para el funcionamiento en UltraStar. Primero, cargamos el archivo de audio mediante librosa y obtenemos los compases por minuto (BPM) del audio. Después, definimos un margen en segundos de corchea (1/8) relativo al BPM leído anteriormente para definir finales de frase. Después, definimos los valores que posteriormente se introducirán en la cabecera.

Una vez hecho esto, realizamos los últimos procesados de audio. Primero, pasamos las notas MIDI a notas de UltraStar y eliminamos las columnas de *pitch_bend*. Después, para definir los finales de frase, silenciamos las partes del audio en las que no se ha detectado voz y, a partir de esos silencios, detectamos los finales de frase y los eliminamos de los datos a leer.

Una vez hecho todo esto, abrimos el archivo UltraStar y empezamos a escribir. Primero, se escribe la cabecera y después las notas mediante un proceso iterativo. El formato que siguen las notas se define en el Anexo II. Si el tiempo entre notas supera el margen definido o llevamos 12 palabras, se considera un final de frase y lo indicamos. Se

han elegido 12 palabras para evitar tener frases demasiado largas que no se vean correctamente en el juego. Una vez terminado, se indica el final de archivo, se comprime el texto junto con los audios y se envía al usuario.

3.7.8 Proceso de video:

Este proceso solo se realiza cuando el usuario pide un vídeo. Primero cargamos el archivo de audio mediante librosa para obtener la duración en segundos del video. El vídeo será un archivo .mp4 con la duración del archivo de audio, el audio será el instrumental obtenido por demucs y el fondo será la letra sobre un fondo en negro. Como un .mp4, al fin y al cabo, es un archivo contenedor, podemos trabajar con cada parte por separado y luego juntarlos.

Para ello, la dependencia MoviePy pone a nuestra disposición objetos Clip para cada parte del video. A continuación, se detallan los procedimientos de cada parte del video:

- Fondo: Se define un fondo negro por medio de ColorClip.
- Texto: Se crea un diccionario con las marcas de tiempo y el texto y se utiliza la librería Pandas para definir un DataFrame. Luego, se crea un clip TextClip en el que definimos las características del texto para que esté en el centro de la pantalla en blanco con bordes negros para resaltado. Con ambos objetos, se define un SubtitlesClip, que define subtítulos en pantalla.
- Audio: Se usa un AudioFileClip con el instrumental obtenido mediante Demucs

Todos estos clips se juntan en un CompositeVideoClip y se escribe a un archivo de video, proceso que dura unos minutos. El archivo de video resultante se envía al usuario.

4 RESULTADOS

Para comprobar el correcto funcionamiento de la aplicación, se han hecho una serie de pruebas de uso que se comentarán en los siguientes apartados.

4.1 Comprobación del manejo correcto a nivel de usuario

En este apartado, se encuentran las pruebas directamente relacionadas con las funciones principales de la aplicación web en manos de un usuario cliente que maneje con éxito la aplicación.

4.1.1 Conversión de un archivo de audio a un archivo UltraStar

Para realizar esta prueba, primero desplegamos la aplicación de conversión y la interfaz de usuario en un ordenador y realizaremos una conexión desde un dispositivo externo para comprobar que el servicio es accesible desde otros dispositivos:

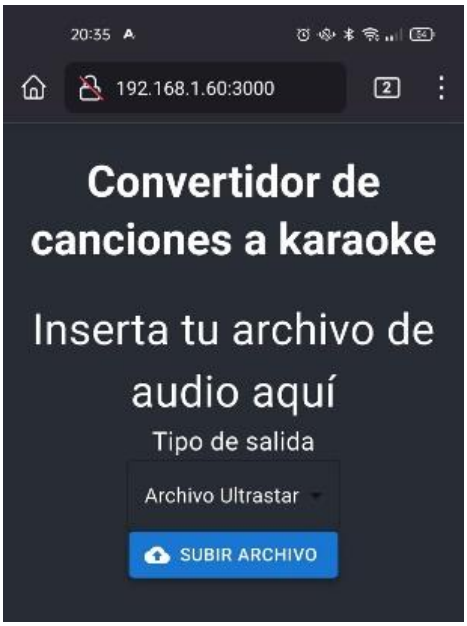

Ilustración 10: Dispositivo a partir del cual se ha realizado la conexión a la interfaz desplegada

Ahora, realizamos un envío de un archivo de audio por medio del botón provisto al usuario. El botón está configurado para que el usuario solo pueda elegir archivos de audio. Al realizarlo, el botón indicado para subir la información se deshabilita.

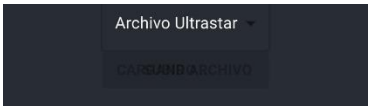

Ilustración 11: Botón de subida de archivo deshabilitado

Desde la consola de comandos donde hemos iniciado la aplicación de conversión, podemos comprobar que el servidor ha recibido con éxito el audio introducido. Para la prueba, hemos utilizado la canción *Yo soy aquél* de Raphael. La consola de comandos nos permite comprobar desde el lado del servidor que la transcripción está en proceso:

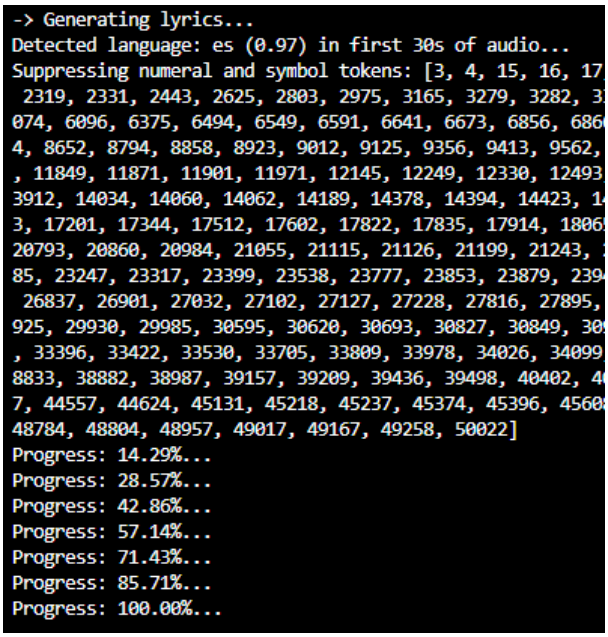

Ilustración 12: Resultado de la terminal donde se ha desplegado el servicio

El proceso de conversión, con el servidor en un hardware de un ordenador portátil personal y usando el modelo *large-v3* de Whisper, duró unos 20 minutos (2-3 si ya hicimos la conversión de karaoke anteriormente). Una vez terminado, el archivo comprimido aparecerá como una descarga automáticamente:

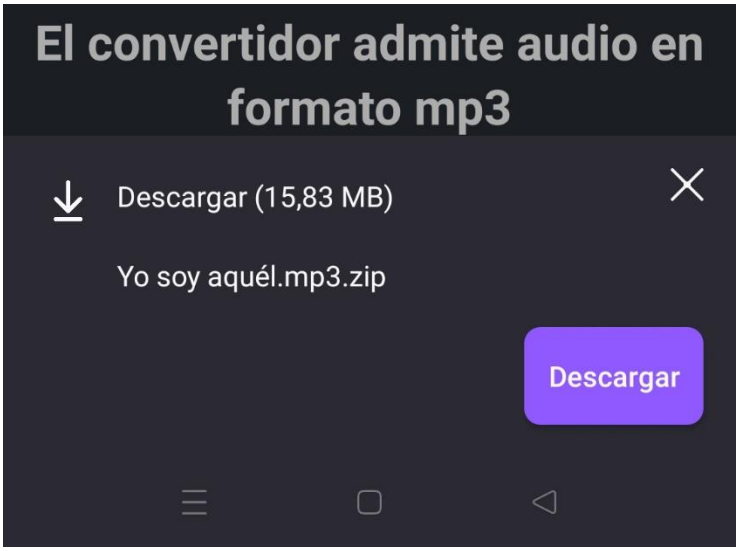

Ilustración 13: Notificación de navegador informando de una descarga de archivo.

Analizando el archivo comprimido, obtenemos los siguientes archivos:

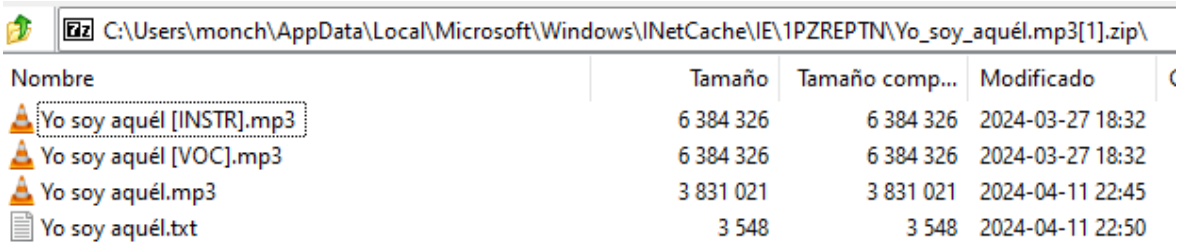

Ilustración 14: Contenido del archivo comprimido

Los dos primeros archivos son el instrumental y el vocal del karaoke, seguidos del audio introducido. Se ha decidido introducirlos porque, en versiones futuras de UltraStar, se planea incluir la posibilidad de modificar de manera gradual la voz en el audio del karaoke, y para ello se necesita ambas versiones. Sin embargo, por lo pronto, se utilizará solo la instrumental como el audio por defecto.

El archivo de texto plano contiene la información necesaria en el formato indicado en la página oficial de UltraStar.

```
#VERSION:1.1.0
#ARTIST:Yo soy aquél
#TITLE:Yo soy aquél
#LANGUAGE:Spanish
#MP3:Yo soy aquél [INSTR].mp3
#AUDIO:Yo soy aquél [INSTR].mp3
#VOCALS:Yo soy aquél [VOC].mp3
#INSTRUMENTAL:Yo soy aquél [INSTR].mp3
#VIDEO:None
#BPM:393.26
#GAP:16217
#CREATOR:Ramon Elbal Ruiz
#COMMENT:Special thanks to Ultrasinger for the format | Number of notes: 305
: 0 4 4 Yo
: 9 4 5 soy
- 13
: 21 4 5 a
: 25 4 5 quel
- 29
: 41 3 9 que
- 44
: 51 4 9 cada
: 55 12 9 no
: 70 8 10 che
- 78
: 88 4 10 te
: 94 6 12 per
: 108 6 12 si
: 116 4 8 gue
- 121
: 163 4 10 Yo
: 173 5 10 soy
: 185 4 10 a
: 189 4 10 quel
- 192
: 209 3 9 que
: 214 11 9 por
: 227 5 10 que
: 232 4 10 rer
: 236 4 10 te
: 247 7 11 ~
- 253
: 265 4 12 ya
: 272 5 14 no
: 277 4 14 vive
: 285 2 9 El
: 288 4 9 que
: 293 7 10 te
: 300 9 10 espera,
- 308
```

Ilustración 15: Archivo de texto incluido en el UltraStar

Las líneas que empiezan con almohadillas contienen metadatos. El resto de las líneas indican los datos de las notas que se verán en el karaoke. Para comprobar que los elementos obtenidos funcionan en UltraStar, utilizaremos el software UltraStar WorldParty.

Introducimos los elementos recibidos en una carpeta dentro del directorio de canciones. El formato y el procedimiento para instalarlos se explica con más detalle en el anexo II.

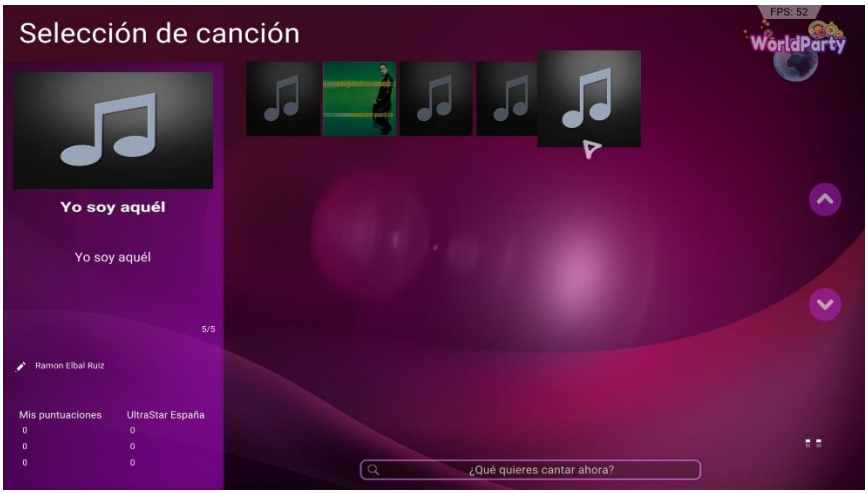

Ilustración 16: Selector de canciones de UltraStar WorldParty

Como podemos observar, la canción aparece en el software de karaoke y se puede jugar. Comprobamos que la canción introducida es jugable sin problemas.

Ilustración 17: Imagen de una partida de UltraStar con el archivo recibido

4.1.2 Comparación de la calidad del karaoke con uno existente

Una vez verificado que el proceso funciona, comprobaremos si la calidad de la transcripción es comparable a la de un karaoke de UltraStar. La metodología es sencilla; desde el catálogo de canciones de UltraStar Worldparty, descargamos una canción, preferentemente una que tenga una buena calificación. Como *Yo Soy Aquel* solo tiene una canción sin valorar, y por tanto no podemos estar seguros de su calidad, elegimos *La del pirata cojo* de Joaquín Sabina y Joan Manuel Serrat:

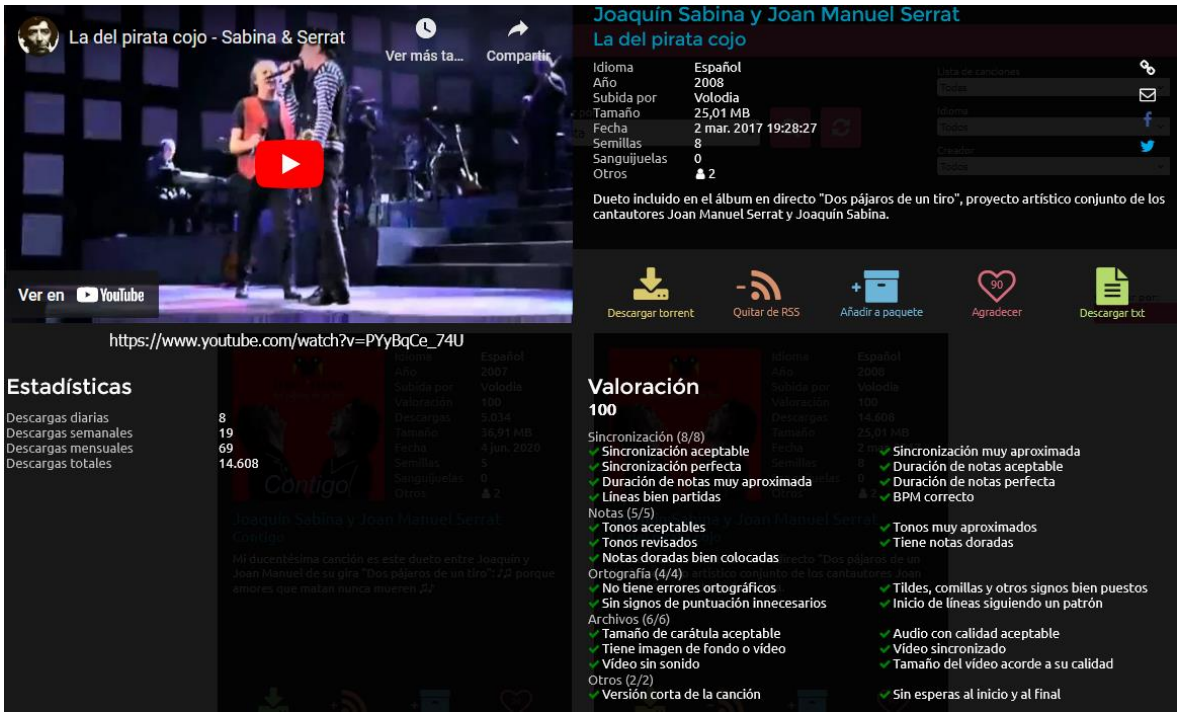

Ilustración 18: Canción elegida como referencia

Una vez elegida la canción, desde nuestra aplicación sacamos un karaoke de UltraStar de exactamente la misma canción y comparamos ambas versiones. Primero, la versión descargada:

Ilustración 19: Canción descargada de Ultrastar

A continuación, se mostrará el resultado de la misma línea en el archivo obtenido por la aplicación:

Ilustración 20: Canción obtenida de la aplicación

Si bien la transcripción logra una calidad comparable y las notas son parecidas, podemos notar ciertas diferencias en el resultado, que son evidentes y debido a las limitaciones que tiene este proceso automatizado en comparación con un proceso humano:

- La separación de frases en la aplicación depende de la separación entre palabras, con un límite introducido de más de 30 sílabas por frase para evitar frases muy largas. El valor que usa nuestra aplicación para detectar un final de frase es 1/8 del BPM de la canción detectado por medio de librosa, que es un valor útil, pero no infalible y por lo tanto vulnerable a falsos positivos y negativos, como la existencia de más saltos de línea de lo normal.
- Whisper separa palabra a palabra, no siendo posible realizar una separación a nivel de sílaba. Para solucionar esta limitación, utilizamos un silabeador y asociamos las sílabas a notas equivalentes en tiempo, con añadido de notas si es necesario. Sin embargo, este proceso añade imprecisión a las notas resultantes, hace que las alucinaciones sean más problemáticas y no tiene en cuenta las sinalefas.
- Whisper funciona peor en escenarios con más de un hablante, especialmente si tienen rangos de voz diferentes.

Teniendo estas limitaciones en mente, podemos considerar que la calidad ofrecida de manera automática por la aplicación es aceptable. La letra coincide con la de la canción salvo errores de transcripción poco notables, está generalmente bien sincronizada con la música y las notas tienen un pitch que concuerda con la parte de la canción que representan. Sin embargo, la separación entre frases es menos precisa y el algoritmo abusa de los apóstrofos en ocasiones.

4.1.3 Conversión de un archivo de audio a un archivo de video

Esta prueba es muy similar al proceso de conversión realizado anteriormente, pero utilizando la opción de recibir el vídeo en un formato de video con fondo negro y letras en pantalla, similar a los videos de *karaoke* con letras. Para permitir la obtención en este formato, basta con cambiar la opción en el selector que indica el tipo de salida. Después, es cuestión de subir el archivo del mismo modo. Utilizaremos el mismo archivo de audio usado en la anterior práctica.

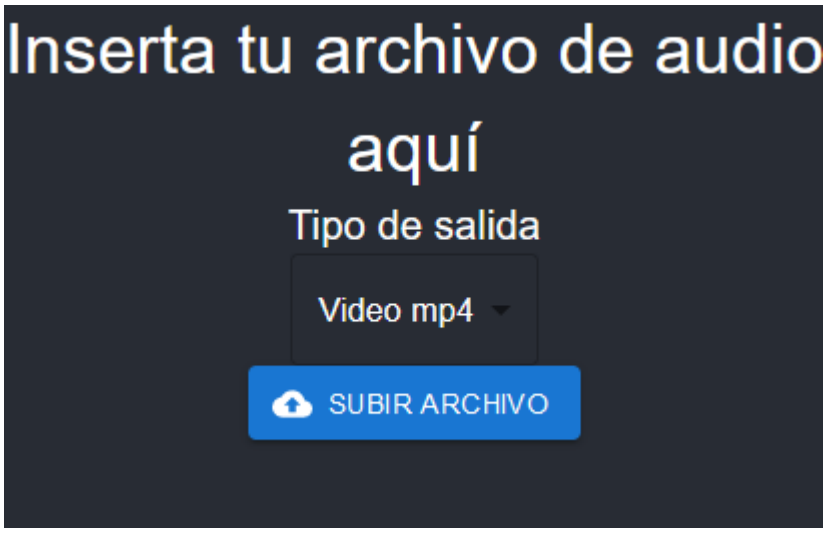

Ilustración 21: Interfaz de usuario con la opción cambiada

Tras unos minutos, el resultado es un video en formato mp4 con el mismo nombre, lo que se espera del archivo introducido en esta modalidad.

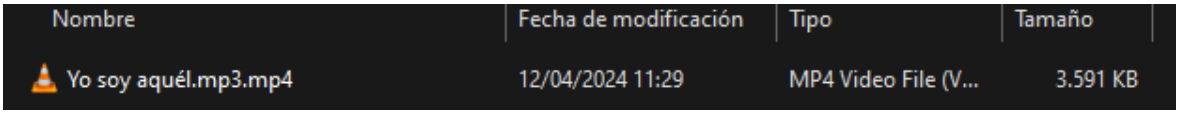

Ilustración 22: Video obtenido tras el proceso

El contenido del video es la letra en color blanco sobre fondo negro con el instrumental de fondo.

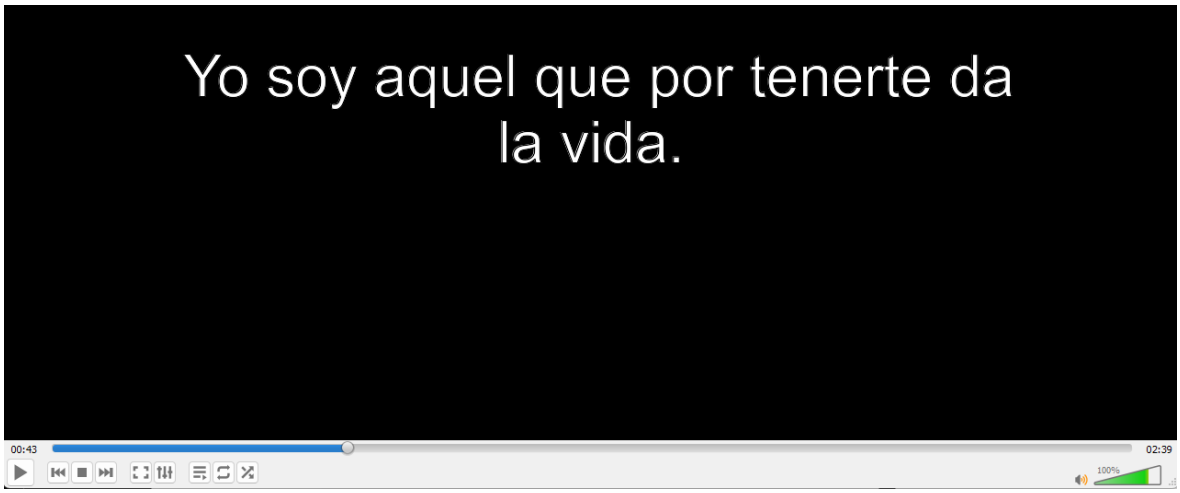

Ilustración 23: Video obtenido siendo reproducido

4.2 Comprobación del manejo correcto de situaciones especiales:

Este apartado se dedicará a pruebas que simulen un uso menos idóneo de la aplicación, para asegurarse de que el servicio no colapsa cuando se utiliza de manera no debida.

4.2.1 Archivo de audio no en formato mp3

La mayoría de los modelos empleados en esta aplicación están optimizados para funcionar con archivos de audio mp3, que es el formato más extendido. Sin embargo, sería interesante ver cómo funciona al enviar archivos de audio en otros formatos que, si bien no son tan extendidos, se siguen utilizando en la actualidad, haciendo hincapié en los formatos con los que UltraStar es compatible.

Se han verificado bajo las mismas condiciones que en el caso de uso correcto, y se han confeccionado los resultados en la siguiente tabla para proveer la información de forma más sencilla y menos redundante:

Formato de audio	Archivo UltraStar	Video mp4
WAV	El formato funciona correctamente	
OGG	El formato funciona correctamente	
FLAC	Funciona, pero es propenso a dar errores de tamaño de archivo y no se recomienda usarlo porque UltraStar no es compatible	

Los modelos no parecen tener problema con formatos de audio, siempre y cuando sean compatibles con ffmpeg, algo no cuestionable en un formato de audio de uso común.

4.2.2 Archivo que no es de audio

Normalmente el botón está diseñado para que el usuario solo suba archivos de audio utilizando las opciones provistas por React. Sin embargo, puede ser que el usuario haga caso omiso y suba un archivo que no sea de audio. Enviar un archivo de estas características causará un error que es detectado por la aplicación y no causa un colapso del servicio.

4.2.3 Archivo superior a 25 MB

Whisper, y por extensión soundfile, librosa, etc… tienen un límite de tamaño de archivo de 25 MB. Este límite impuesto no debería ser un problema para la mayoría de los archivos de audio, menos formatos sin pérdidas como FLAC o audios de muy larga duración. Subir un archivo por encima de este límite dará un error en medio del proceso que es detectado.

4.2.4 El usuario abandona la página durante el proceso

Como cabe de esperar, si el usuario abandona la página, no recibirá el resultado de la conversión, ya que la respuesta no se llega a enviar.

4.2.5 Se recibe más de una solicitud a la vez

Como es una aplicación web, se espera que la aplicación no colapse al recibir solicitudes incluso cuando está realizando los procesos para atender otras. Normalmente FastAPI ejecutaría los procesos de manera asíncrona, pero Tensorflow impide ejecutar múltiples conversiones a la vez en la misma GPU. Por tanto, como alternativa, se ha optado por bloquear el hilo mientras se está ejecutando la conversión y desbloquearlo cuando se complete para que otro proceso lo use. De esta forma, se evitan errores derivados de ejecutar múltiples procesos dependientes de Tensorflow en la misma GPU.

En definitiva, sería como poner las solicitudes en una cola, lo cual no es la opción más idónea para un sistema de estas características, pero previene el problema descrito anteriormente. Para probar que el servidor funciona al recibir múltiples solicitudes, se ha dispuesto un escenario en el que se enviará desde 4 instancias de la aplicación web, un archivo de audio diferente al servidor al mismo tiempo:

Ilustración 24: Conversión iniciada desde múltiples ventanas, cada una con un archivo diferente

El resultado esperado es que cada ventana con el tiempo reciba su respectiva solicitud que le ha mandado al servidor. En la ventana de comandos donde se ejecuta el programa, se muestra cómo cada proceso se realiza uno detrás de otro, como si estuvieran en una cola:

```
  Creating note events...
  ✿ Saved to C:\Users\monch\Downloads\TFGteleco\lyrics-bot-main\src\midi\vocals_basic_pitch.csv
     start_time_s  end_time_s  pitch_midi  velocity  pitch_ultrastar
373      0.150930    0.220590          70        43             22.0
772      0.162540    0.835918          29        71            -19.0
766      0.777868    1.532517          22        65            -26.0
761      1.219048    1.613787          33        70            -15.0
308      1.439637    1.544127          57        55              9.0
..            ...         ...         ...       ...              ...
868    235.450911  235.601840          35        43            -13.0
1      235.450911  235.520571          62        66             14.0
871    236.019800  236.089459          67        44             19.0
850    236.066239  236.193949          62        46             14.0
0      236.368099  236.450653          66        66             18.0

[875 rows x 5 columns]
[UltraSinger] Mute audio parts with no singing
[UltraSinger] Removing silent parts from transcription data
Aún no
My current directory is C:\Users\monch\Downloads\TFGteleco\lyrics-bot-main\src
INFO:     192.168.1.60:62913 - "POST /video HTTP/1.1" 200 OK
-> Generating lyrics...
C:\Users\monch\Downloads\TFGteleco\lyrics-bot-main\src\results\Joaqun Sabina La del Pirata Cojo\Jo
ata Cojo.mp3
Detecting language using up to the first 30 seconds. Use `--language` to specify the language
Detected language: Spanish
```

Ilustración 25: La primera conversión termina y la segunda se inicia justo después

Al haberse hecho las solicitudes en ventanas del mismo navegador, la descarga aparece en las 4, pero se demuestra que el sistema puede atender con éxito múltiples solicitudes. En la siguiente imagen, se observa que las 4 solicitudes han sido respondidas con éxito:

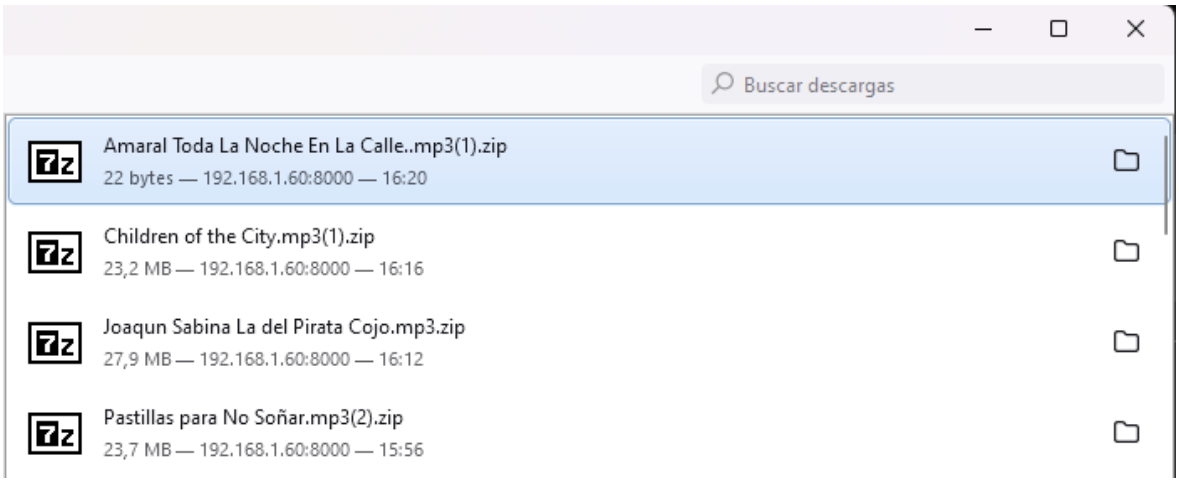

Ilustración 26: Descargas realizadas con éxito

5 CONCLUSIONES Y LÍNEAS FUTURAS DE DESARROLLO

En este apartado, se determinará si el trabajo presentado cumple con los objetivos propuestos y se propondrán mejoras o nuevas funcionalidades que el autor considere.

Para empezar, se considera que el trabajo cumple con éxito los objetivos propuestos en el TFG. La aplicación permite al usuario enviar un archivo de audio y recibir un paquete de archivos que puedan ser introducidos en un software de karaoke. El proceso de conversión conlleva separación de voz e instrumental, transcripción y sincronización de las letras, detección de pitch, así como post-procesado mediante alineamiento forzado y silabeado, todos ellos realizados con modelos de redes neuronales preentrenados. El intercambio de archivos se realiza por medio de un servicio web con una interfaz de usuario.

Además del objetivo propuesto, se ha llevado a cabo, a modo de meta complementaria, la implementación de la opción de convertir el archivo de audio en un video con el instrumental y los subtítulos en pantalla, ya que el alumno ha observado que es un formato que también es popular debido a su relativa conveniencia.

Sin embargo, el trabajo tiene ciertos aspectos que mejorar. La interfaz de usuario, si bien cumple con el propósito de la aplicación, tiene aún que pulirse en cuando a diseño y robustez, así como proporcionar información del proceso, aunque sea poco precisa, para mantener al usuario en la página.

Además, se ha observado que el proceso de conversión tiene una gran duración, mucho más de lo que uno puede esperar de una plataforma web. Por un lado, se puede achacar al hardware en el que se ha probado la mayor parte del tiempo, un ordenador portátil de alta gama, pero de uso personal, y por lo tanto no optimizado para desplegar servidores que utilicen varios modelos de inteligencia artificial en materia de procesado de audio. Por otra parte, los procesos necesarios son inherentemente largos, y más aun dependiendo del formato de audio y su longitud y los modelos que queramos utilizar. Sin ir más lejos, se ha observado que más de la mitad del proceso de conversión se ha dedicado a la extracción del instrumental, seguido de la transcripción de la letra y la confección del video (en su caso). Esto puede ser un problema ya que se espera que el usuario mantenga la ventana del navegador abierta esperando la respuesta del servidor. A esto hay que añadir que actualmente la aplicación no se ha escalado para llevar a cabo múltiples conversiones a la vez, sino que se utiliza un sistema de colas debido a limitaciones del proceso. Es decir, en el caso de que haya más de un usuario queriendo hacer conversiones, el último usuario en subir algo debe tener la ventana abierta hasta que algo ocurra.

Además, si bien los modelos son extraordinariamente precisos y los resultados de las pruebas realizadas han sido satisfactorios, no son perfectos. Al igual que en las transcripciones reales, siempre puede haber errores, aunque sean relativamente minúsculos, de sincronización, de letra, etc… Estos errores no son percibidos como tal por los propios modelos, por lo que recae en un post-procesado más exhaustivo el minimizarlos o en el juicio humano el corregirlos. A esto hay que añadir que Whisper, por diseño, solo devuelve marcas de tiempo a nivel de palabra, no de sílaba, lo que nos obliga a utilizar post-procesado para obtener marcas de tiempo a nivel de sílaba, tal y como se espera de un karaoke. Este post-procesado no es perfecto y es más propenso a errores que el resto de los procesos juntos.

Por último, no todos los sistemas de análisis de texto y voz son compatibles con todos los idiomas. Whisper de por sí no tiene muchos problemas de compatibilidad de idiomas, pero el alineamiento forzado de WhisperX es relativamente más limitado a menos que se añadan modelos en otros idiomas al programa y los analizadores de sílabas están limitados por los diccionarios de los que se quiera disponer. A cada algoritmo más limitado en cuanto a idiomas compatibles, se crea un cuello de botella en cuanto a la compatibilidad de idiomas del sistema en su conjunto. Sin embargo, estas restricciones son superables añadiendo al programa limitante el contenido necesario para aportar compatibilidad, ya sea un modelo de alineamiento o un diccionario.

Respecto a funcionalidades futuras que se podrían implementar serían:

- Un proceso intermedio en el cual se muestre la letra en la interfaz para que el usuario pueda realizar modificaciones antes de usarse en el resto de los procesos.
- Escalar la aplicación para poder realizar múltiples conversiones a la vez, por ejemplo, por medio de contenedores virtuales como Docker.
- Permitir al usuario añadir elementos cosméticos al archivo UltraStar (portada, vídeo, metadatos, etc…)
- Mejorar la compatibilidad con múltiples idiomas mediante la implementación de diccionarios o modelos para otros idiomas en los programas ya existentes.

6 ANEXOS

6.1 Anexo I: Manual técnico y de despliegue

Este anexo describirá la información necesaria para construir el entorno virtual con las dependencias necesarias, desplegar la aplicación web y solucionar posibles problemas.

6.1.1 Requerimientos de despliegue

Para el despliegue de la aplicación se requieren los siguientes programas adicionales instalados en el dispositivo:

- La última versión de Node.js, cuya instalación se puede realizar descargándose el instalador desde la página oficial.
- La última versión de Python, recomendable añadirla al PATH del sistema operativo.
- Conda o Mamba, para manejar entornos virtuales de Python.
- ffmpeg en el PATH del sistema operativo o en el mismo directorio que el script.
- El software ImageMagick o sus dependencias de Python equivalentes.

6.1.2 Instalar CUDA

Antes de instalar el entorno virtual, tenemos que instalarnos el toolkit de CUDA si no lo tenemos antes. La última versión se puede encontrar en https://developer.nvidia.com/cuda-downloads . Una vez instalado, podemos comprobar que se ha instalado correctamente si tras el comando nvv --version nos sale algo similar a esto:

```
(demucsnew) C:\Users\monch\Downloads\TFGteleco\lyrics-bot-main\src>nvcc --version
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2023 NVIDIA Corporation
Built on Wed_Nov_22_10:30:42_Pacific_Standard_Time_2023
Cuda compilation tools, release 12.3, V12.3.107
Build cuda_12.3.r12.3/compiler.33567101_0
```

Ilustración 27: Resultado del comando nvcc --version

6.1.3 Instalar las dependencias en un entorno virtual

Una vez instalados los programas, el primer paso será instalar las dependencias necesarias. Para casos como este, es muy útil la posibilidad de generar un entorno virtual fácilmente compartible utilizando un archivo. Este entorno viene definido en el archivo *entorno.yml* situado en la carpeta *src* de nuestro proyecto.

La definición contiene las dependencias que se instalarán con la opción de detallar requisitos de versiones de cada dependencia. Si no se detalla una versión, conda utilizará la versión más reciente posible. Se recomienda utilizar las versiones introducidas, para que dé la menor cantidad de problemas posible debido a compatibilidad de dependencias. El

entorno ha sido probado e instalado en Windows, pero puede ser que algunas dependencias no funcionen en Linux.

Para crear el entorno, introduciremos en una consola de comandos ejecutada como administrador el comando `conda env create -f entorno.yml`. Este comando hace que conda determine e instale las dependencias en base a las restricciones de versiones introducidas en el archivo, proceso muy largo que se define como *resolver* el entorno.

Si el entorno está creado con éxito, deberíamos ser capaces de utilizar ese entorno mediante el comando `conda activate demucs`.

En caso de que no se instale correctamente, las dependencias que hacen falta instalar se pueden hacer de forma manual utilizando la consola. En la siguiente lista se detallan las dependencias que hace falta instalar por medio de sus respectivos instaladores de paquetes para que funcione el programa. Si no se menciona versión, se considera que es la versión más nueva que permita el instalador respectivo:

Conda:

Canales:

- pytorch
- nvidia
- conda-forge
- defaults

Dependencias:

- python>=3.9.18
- fastapi>=0.103.0
- ffmpeg>=4.2
- pandas>=2.2.1
- pip >= 23.2
- langcodes>=3.3.0
- pytorch
- torchvision
- torchaudio
- pytorch-cuda==[la versión correspondiente al CUDA toolkit]

Pip:

- basic-pitch[tf] (o basic-pitch y tf-runtime por separado)
- moviepy>=1.0.3
- config>=0.5.1
- demucs>=4.0.1
- openai-whisper

- librosa==0.10.1 obligatorio porque versiones posteriores cambian el formato de la salida
- uvicorn>=0.30.1
- pydub>=0.25.1
- pipreqs
- prosodic
- pyverse
- python-multipart>=0.0.9
- scipy==1.12.0
- tensorflow
- ipython==8.12.3
- whisperx>=3.1.4
- mido
- language-data
- numpy~=1.26.4

Estas dependencias, junto con los pasos siguientes, deberían garantizar que se tiene todo lo necesario para ejecutar el programa.

6.1.4 Instalar pytorch

Si bien el proceso anterior instala la mayoría de las dependencias sin problemas, es muy posible que tengamos el siguiente error al intentar ejecutar el script:

```
AssertionError: Torch not compiled with CUDA enabled
```

Si esto ocurre, nos vamos a la página web https://pytorch.org/get-started/locally/ y seleccionamos la versión estable y la versión de CUDA más reciente para la cual hay soporte y el gestor de dependencias que queramos (pip o conda). Hay que tener en cuenta que si usamos CUDA 12.4 o superior, hay un método de instalación diferente. Más información en la página web:

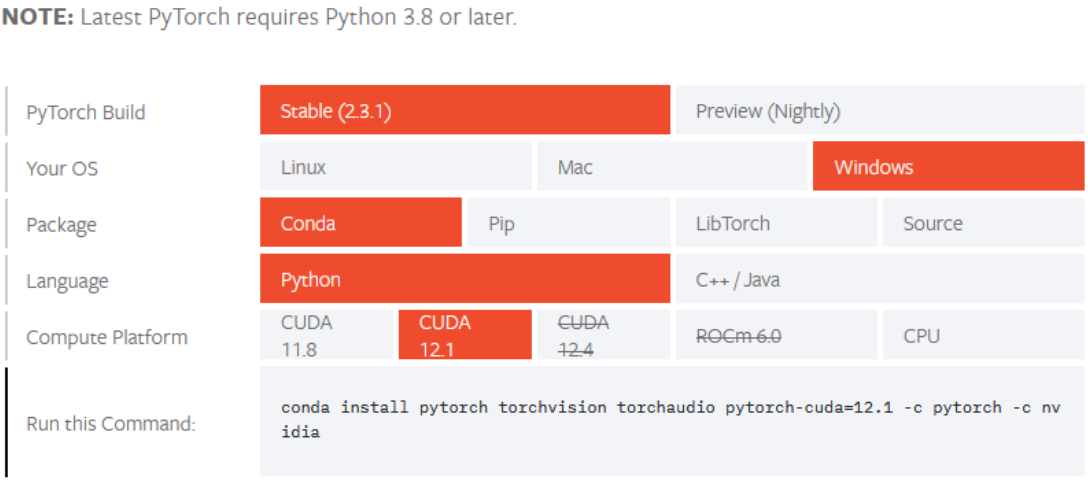

Ilustración 28: Página web que permite ver el comando necesario para instalar dependencias

Pegamos ese comando (con --upgrade) en nuestra terminal con el entorno activado y se instalarán las dependencias.

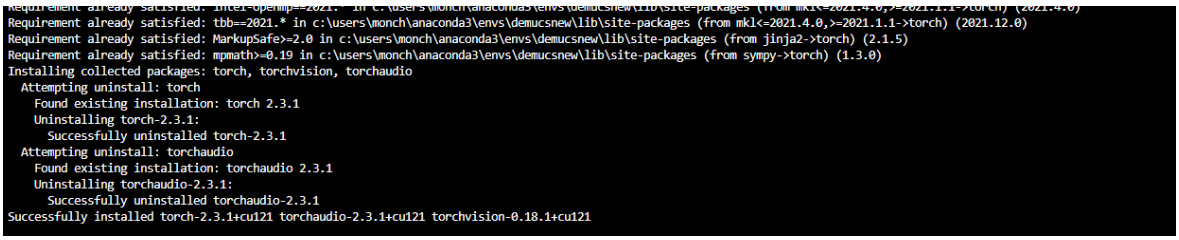

Ilustración 29: Instalación de las dependencias de Pytorch

6.1.5 Despliegue del servidor

Una vez generado el entorno con éxito e instalado todas las dependencias, podemos iniciar el despliegue de nuestra aplicación de conversión. Para ello, mientras estemos en el entorno creado desde una terminal con permisos de administrador, utilizamos el comando `python bot.py`, que ejecuta el script de la aplicación de conversión. Si el entorno lo ejecuta sin problemas, deberíamos ver en la terminal el despliegue de la aplicación:

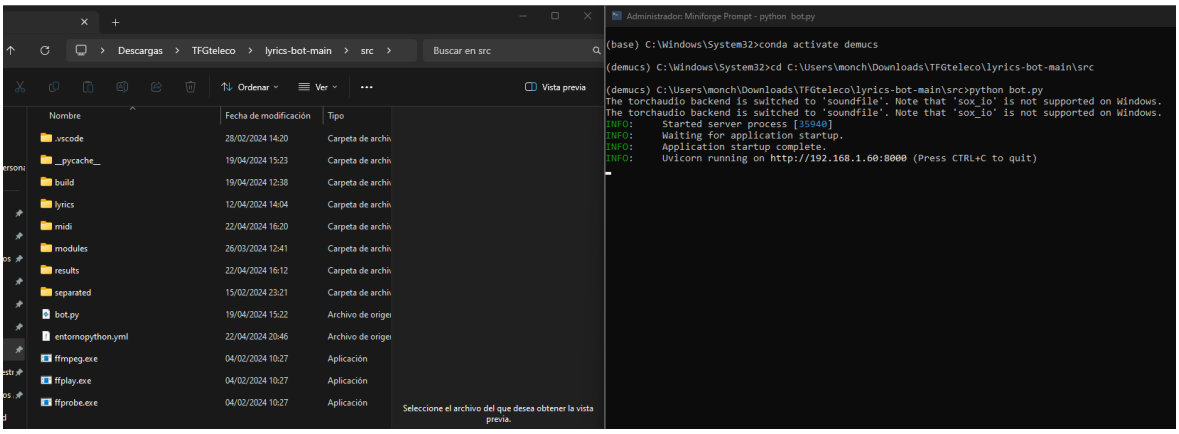

Ilustración 30: Despliegue de la aplicación de conversión

Esto despliega el servidor web en la dirección y puerto indicados. Ahora, para que un dispositivo se conecte, solo ha de introducir la dirección indicada en un navegador web y debería de poder verse la interfaz de usuario:

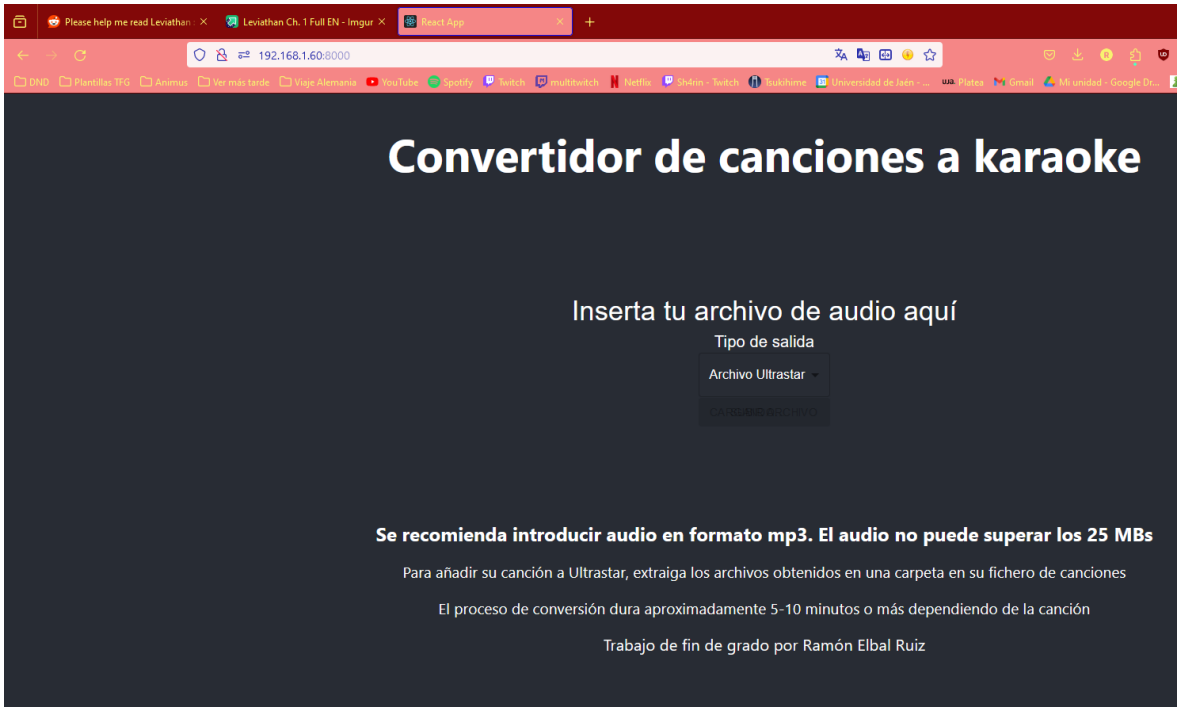

Ilustración 31: Navegador web mostrando la interfaz de usuario

6.1.6 Solución de problemas

A continuación, se describen algunos problemas que pueden surgir al utilizar la aplicación y cómo resolverlos:

6.1.6.1 La función que genera el video devuelve "TypeError: must be a real number, not NoneType"

Reinstala moviepy y decorator e instala moviepy otra vez mediante los siguientes comandos:

```
pip uninstall moviepy decorator
pip install moviepy
```

6.1.6.2 Moviepy no puede detectar ImageMagick

Primero, verifica si el software ImageMagick (o las dependencias de Python equivalentes) ha sido instalado correctamente. Si está instalado, en el directorio de instalación de ese programa encontrará un documento llamado policy.xml.

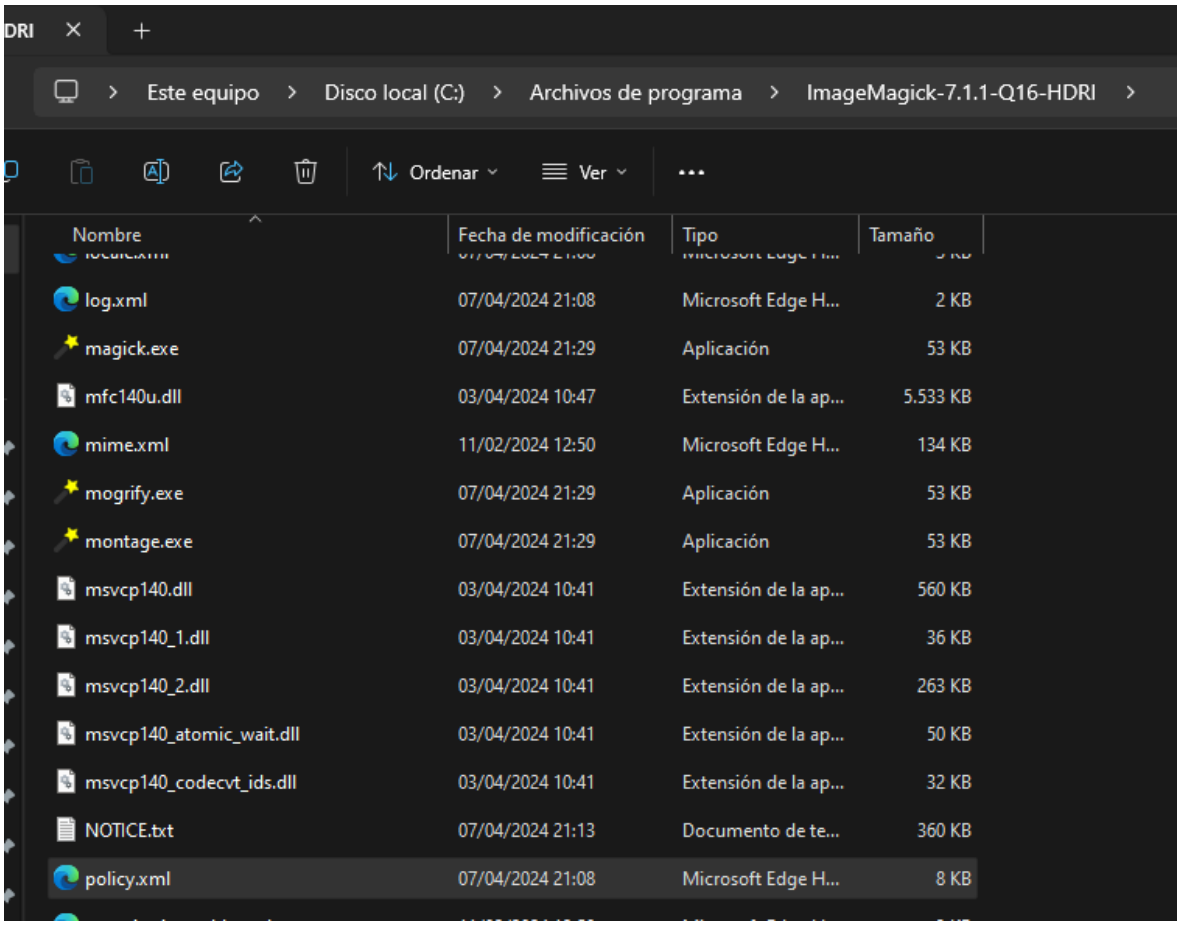

Ilustración 32: Carpeta de instalación de ImageMagick

Abra un editor de texto y comente añadiendo un <!-- la siguiente política:

```
<policy domain="path" rights="none" pattern="@*" />
```

El resultado debería ser como la siguiente imagen:

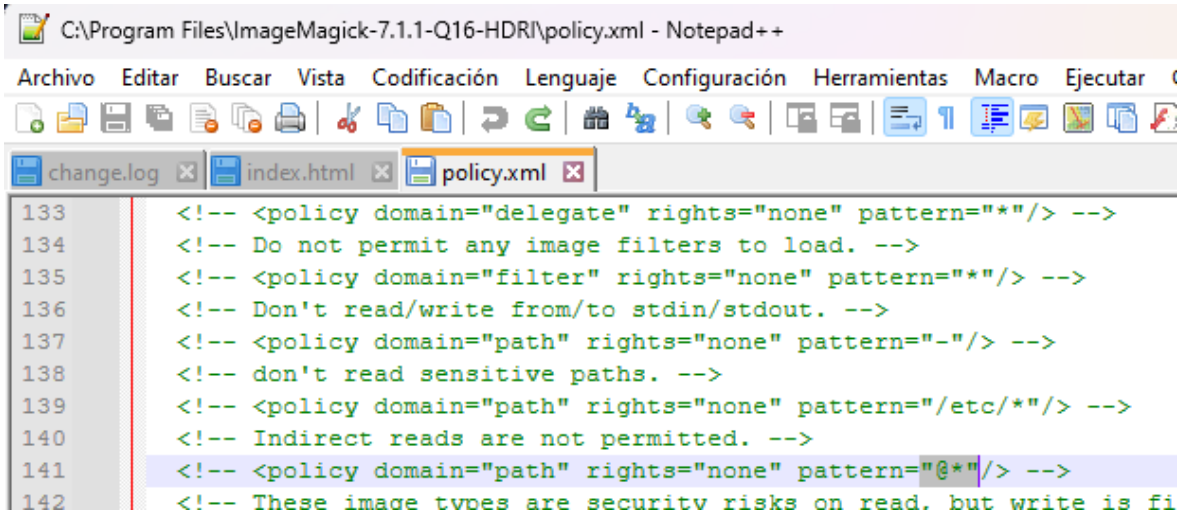

Ilustración 33: Archivo policy.xml

6.1.6.3 OSError: cannot load library 'libsndfile.dll': error 0x7e

Error común de pysoundfile que le impide detectar la librería en cuestión. Desinstale la dependencia soundfile y vuélvela a instalar.

```
pip uninstall soundfile
pip install soundfile
```

6.1.6.4 TypeError: TranscriptionOptions.__new__() missing 3 required positional arguments: 'max_new_tokens', 'clip_timestamps', and 'hallucination_silence_threshold'

Bug causado en ciertas versions de WhisperX. Añade las variables de entrada a la definición del método en cuestión o reinstala WhisperX para que lea la función de transcripción correctamente:

```
pip uninstall whisperx
pip install whisperx
```

6.1.6.5 La aplicación se queda en el paso de detección de eventos MIDI

La aplicación necesita acceder a resultados de pasos anteriores que se guardan localmente, pero para ello necesita permisos de administrador. Para que esto no ocurra, es imperativo que la terminal que ejecuta el script se ejecute como administrador.

6.1.6.6 Out of Memory:

Alguno de los modelos utilizados no dispone de memoria suficiente como para ejecutar un proceso. Esto se puede deber a que la GPU del equipo no es lo suficientemente potente (o la CPU en el caso de que se use, pero es poco recomendable). Para limitar el uso de memoria, se puede utilizar un modelo menos potente en el proceso afectado o usar atributos del modelo en cuestión para limitar su uso de memoria, por ejemplo, --segment [duración del segmento] en el caso de Demucs para limitar la longitud de los segmentos.

6.2 Anexo II: Formato y uso de archivos UltraStar

Este anexo explicará el formato de UltraStar, el software de karaoke que se ha decidido soportar, así como la forma de utilizar los archivos obtenidos en UltraStar. En cierta medida, se puede tratar como un manual a nivel de usuario. En UltraStar, todas las canciones de karaoke están introducidas en la carpeta *songs* dentro del directorio de instalación, cada una ocupando una subcarpeta. Para acceder a dicho directorio, podemos seguir la ruta que aparece o escribir *Canciones* en el buscador una vez instalada la aplicación, lo que mostrará un acceso directo generado al instalar el programa:

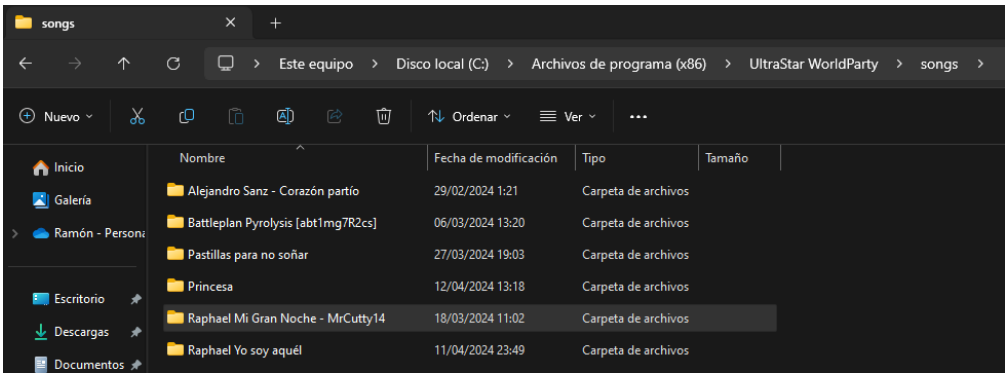

Ilustración 34: Directorio de canciones de UltraStar

Para añadir una canción de UltraStar, hacen falta como mínimo dos ficheros: La canción (en formato mp3, wav u ogg) y el fichero de texto plano codificado en UTF-8 donde se introducirán los metadatos e información sobre las notas. Ambos se incluyen en el fichero UltraStar proporcionado por la aplicación.

También se pueden introducir archivos adicionales, como un fondo, un vídeo o una versión instrumental y una vocal que se usen para juegos donde puedas cambiar el volumen del cantante (esto último también viene en el paquete). Han de introducirse en la misma carpeta y nombrarse con la etiqueta respectiva. A continuación, se pasará a explicar el formato del archivo de texto. La parte inicial del archivo de texto serán los atributos, que siempre seguirán el formato *#ATRIBUTO*:*Valor*. Con algunas excepciones, la mayoría son opcionales (32). A continuación, se explicarán los que se han utilizado en la canción:

- **VERSION:** Indica la versión de UltraStar para la que fue diseñada la canción. Obligatorio
- **TITLE:** Título de la canción. Obligatorio
- **ARTIST:** Artista de la canción. Obligatorio
- **MP3:** Nombre del archivo de audio que se oirá. Obligatorio, pero será depreciado en 2025 y sustituido por **AUDIO**, que tiene la misma función. En el archivo de texto de la aplicación, se usarán las dos para que la aplicación esté preparada para el futuro.
- **BPM:** Velocidad de la canción en negras (quarter-notes), por lo que idealmente, el valor debe ser 4 veces el BPM real de la canción. Obligatorio
- **GAP:** Tiempo en milisegundos para el inicio de la letra con respecto al inicio de la canción.
- **VOCALS:** Archivo de audio con la versión *acapella* de la canción. El archivo de audio ha de terminar en [VOC].
- **INSTRUMENTAL:** Archivo de audio con la versión instrumental de la canción. El archivo de audio ha de terminar en [INSTR].
- **CREATOR:** Creador o creadores del archivo de texto

- **LANGUAGE:** Idioma de la canción. Usado para filtrar canciones por idioma
- **COMMENT:** Información adicional que no se utiliza dentro de la aplicación.

Después de los atributos, se indican las notas, la parte más importante del juego. Cada nota ocupa una línea con el siguiente formato:

NoteType StartBeat Length Pitch Text

- **NoteType:** Tipo de nota denotado con un símbolo. Tenemos notas normales, notas doradas (dan el doble de puntos), notas *freestyle* (no dan puntos) y las mismas variantes para notas de Rap. En nuestro caso, todas las notas serán normales.
- **StartBeat:** El compás en el que comienza la nota, relativo a los campos BPM y GAP.
- **Length:** Longitud de la nota, también es relativa a BPM y GAP.
- **Pitch:** Tono de la nota. 0 corresponde a la nota C4 o Do central.
- **Text:** El texto al que corresponde la nota

Para delimitar las frases, se introduce la línea:

- *StartBeat*

Donde *StartBeat* se calcula de la misma forma que en una nota.

Al final del fichero de texto, se colocará una línea con una *E* para indicar el final del archivo.

7 REFERENCIAS

1. **Google Inc.** Vision AI: Herramientas de IA. [En línea] 2 de Noviembre de 2023. [Citado el: 12 de Abril de 2024.] https://cloud.google.com/vision?hl=es.

2. **Singh, Snehai.** Karaoke Market Size, Share, Growth, Global Industry Analysis – 2024 | MRFR. *Karaoke Market Size, Share, Growth, Global Industry Analysis – 2024 | MRFR.* February de 2021.

3. **Microsoft.** Visual Studio Code . Code Editing Redefined. [En línea] 2023. https://code.visualstudio.com/.

4. **Github Inc.** Github: Let's build from here. [En línea] 2023. https://github.com.

5. **Postman Inc.** Postman API Platform | Sign Up for Free. [En línea] 2023. [Citado el: 2 de Abril de 2023.] https://www.postman.com.

6. **Oracle VM VirtualBox. [En línea] Oracle, 9 de Diciembre de 2023. [Citado el: 5 de 01 de 2023.] https://www.virtualbox.org/.**

7. UltraStar España. [En línea] 2023. [Citado el: 7 de Diciembre de 2023.] https://ultrastar-es.org/es.

8. Anaconda | The Operating System for AI. [En línea] 2024. [Citado el: 9 de DIciembre de 2023.] https://www.anaconda.com/.

9. Github Co. mamba-org/mamba: The Fast Cross-Platform Package Manager. [En línea] 2024. [Citado el: 9 de Diciembre de 2023.] https://github.com/mamba-org/mamba.

10. —. conda-forge/miniforge: A conda-forge distribution. [En línea] [Citado el: 9 de Diciembre de 2023.] https://github.com/conda-forge/miniforge.

11. —. rakuri255/UltraSinger: AI based tool to convert vocals lyrics and pitch from music to autogenerate Ultrastar Deluxe, Midi and notes. It automatic tapping, adding text, pitch vocals and creates karaoke files. [En línea] 6 de Mayo de 2024. https://github.com/rakuri255/UltraSinger.

12. @tiangolo. FastAPI. [En línea] 2024. [Citado el: 8 de Enero de 2024.] https://fastapi.tiangolo.com/.

13. tiangolo. Uvicorn. [En línea] 2024. [Citado el: 13 de Febrero de 2024.] https://www.uvicorn.org/.

14. TensorFlow. [En línea] [Citado el: 15 de Febrero de 2024.] https://www.tensorflow.org.

15. ImageMagick Studio, L. L. C. ImageMagick – Mastering Digital Image Alchemy. *ImageMagick – Mastering Digital Image Alchemy.* 2023.

16. Heuser, Ryan, Falk, Josh y Anttila, Arto. quadrismegistus/prosodic: Prosodic: a metrical-phonological parser, written in Python. For English and Finnish,

with flexible language support. *quadrismegistus/prosodic: Prosodic: a metrical-phonological parser, written in Python. For English and Finnish, with flexible language support.* June de 2024.

17. Nebur, Ruben Karlsson. neburnodrog/Pyverse: A syllabification algorithm for Spanish verses in Python. *neburnodrog/Pyverse: A syllabification algorithm for Spanish verses in Python.* June de 2024.

18. librosa. librosa/librosa: Python library for audio and music analysis. [En línea] [Citado el: 28 de Febrero de 2024.] https://github.com/librosa/librosa.

19. Zulko. User Guide — MoviePy 1.0.2 documentation. [En línea] 2017. [Citado el: 9 de Diciembre de 2023.] https://zulko.github.io/moviepy/.

20. Radford, Alec, y otros. Robust Speech Recognition via Large-Scale Weak Supervision. *Robust Speech Recognition via Large-Scale Weak Supervision.* 2022.

21. HuggingFace.co. Open ASR Leaderboard. [En línea] 10 de Febrero de 2024. [Citado el: 21 de Mayo de 2024.] https://huggingface.co/spaces/hf-audio/open_asr_leaderboard.

22. *WhisperX: Time-Accurate Speech Transcription of Long-Form Audio.* Bain, Max, y otros. 2023, INTERSPEECH 2023.

23. Co., Github. *SYSTRAN/faster-whisper: Faster Whisper transcription with CTranslate2.* May 2024.

24. Gondi, Santosh. Wav2Vec2.0 on the Edge: Performance Evaluation. *Wav2Vec2.0 on the Edge: Performance Evaluation.* 2022.

25. Shahin, Mostafa, Epps, Julien y Ahmed, Beena. Phonological Level wav2vec2-based Mispronunciation Detection and Diagnosis Method. *Phonological Level wav2vec2-based Mispronunciation Detection and Diagnosis Method.* 2023.

26. Défossez, Alexandre. Hybrid Spectrogram and Waveform Source Separation. *Hybrid Spectrogram and Waveform Source Separation.* 2022.

27. Spotify AB. Basic Pitch: An open source MIDI converter from Spotify - Demo. [En línea] 2022. [Citado el: 21 de 2 de 2024.] https://basicpitch.spotify.com/.

28. *A Lightweight Instrument-Agnostic Model for Polyphonic Note Transcription and Multipitch Estimation.* Bittner, Rachel M., y otros. Singapore : s.n., 2022. Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP).

29. Meta. React. [En línea] 2023. [Citado el: 31 de Marzo de 2023.] https://react.dev.

30. Stack Overflow. Stack Overflow Developer Survey 2023. [En línea] 2023. [Citado el: 5 de Mayo de 2024.] https://survey.stackoverflow.co/2023/#section-most-popular-technologies-programming-scripting-and-markup-languages.

31. Material UI SAS. MUI: The React component library you always wanted. [En línea] 2023. [Citado el: 12 de Marzo de 2023.] https://mui.com.

32. UltraStar. UltraStar Format. [En línea] LazyFox Design, 21 de Noviembre de 2023. [Citado el: 24 de Febrero de 2023.] https://usdx.eu/format/.