UNIVERSIDAD POLITÉCNICA DE MADRID

E.T.S. DE INGENIERÍA DE SISTEMAS INFORMÁTICOS

PROYECTO FIN DE GRADO

GRADO EN INGENIERÍA DEL SOFTWARE

Desarrollo de una aplicación basada en Inteligencia Artificial para la transcripción avanzada de audio

Autora: Daniel Salgado Infantes

Directora: Víctor Rodríguez Fernández

Madrid, 5 de julio de 2023

Daniel Salgado Infantes

Desarrollo de una aplicación basada en Inteligencia Artificial para la transcripción avanzada de audio
Proyecto Fin de Grado, 5 de julio de 2023
Directora: Víctor Rodríguez Fernández

E.T.S. de Ingeniería de Sistemas Informáticos
Campus Sur UPM, Carretera de Valencia (A-3), km. 7
28031, Madrid, España

Agradecimientos

A todas aquellas personas que contribuyeron de manera invaluable durante todo este proceso académico. En primer lugar, quiero agradecer a mi supervisor, Víctor, por su orientación experta, apoyo constante y valiosas sugerencias a lo largo de todo el proceso. También mostrar mi gratitud a mis compañeros de clase y amigos, quienes brindaron su apoyo moral y compartieron su experiencia, ayudándome a superar todas las dificultades que surgieron en el camino. Además, quiero agradecer a mi familia por su su paciencia y su constante aliento durante este período. Sin su aporte, esta investigación no hubiera sido posible. A todos ellos, mi más profundo agradecimiento.

Resumen

Hoy en día la transcripción de audios a texto se ha convertido en una necesidad esencial ya sea en el ámbito profesional o en el ámbito personal. Sin embargo, encontrar una herramienta de transcripción de audios que muestre el texto en un formato legible, es decir, separando cada intervención de los interlocutores y dividiéndolo en párrafos, resulta realmente complicado, pues no existen herramientas que integren todos estos requisitos a la vez.

Esta es la razón por la que se identificó la necesidad de desarrollar una herramienta que integrase al mismo tiempo los servicios de diarización de interlocutores, transcripción a texto de un audio, y segmentación en párrafos de dicho texto. Para agilizar, mejorar y realizar estos procesos, actualmente existen modelos ya entrenados de Inteligencia Artificial que puede realizar estas funciones, lo que implica que se deben de integrar estos tres modelos.

Para la diarización de interlocutores se ha usado Pyannote.Audio que contiene modelos ya entrenados para la diarización de hablantes. Este modelo devuelve como resultado una lista de cada intervención con el identificador del interlocutor y los milisegundos exactos de inicio de dicha intervención y también los del final de la misma. Otro modelo que se usa es el de Whisper, el modelo de transcripción automático de OpenAI que permite transcribir audios a muchos formatos, incluido el texto. Por último, también se hace uso de GPT-3.5, también de OpenAI, el cual es un modelo de aprendizaje autorregresivo diseñado para generar textos escritos de manera automática. En este caso, utilizando el prompt adecuado se puede llevar a cabo la tarea de la segmentación en párrafos.

Como resultado del desarrollo de esta herramienta se ha conseguido crear un puente entre las tareas de Inteligencia Artificial de diarización, transcripción y segmentación en párrafos de un audio, facilitando el uso de estos modelos de Inteligencia Artificial tanto para el ámbito profesional como para el ámbito personal de los usuarios.

El código fuente se encuentra en un repositorio de GitHub, para que cualquier usuario pueda utilizar esta herramienta y mejorarla o modificarla para que se adapte a sus necesidades.

Palabras clave: Automática; Inteligencia Artificial; Transcripción de Audio; Diarización de Interlocutores; Segmentación en Párrafos

Abstract

Nowadays, audio transcription to text has become an essential need, whether in a professional or personal context. However, finding a transcription tool that displays the text in a readable format, separating each speaker's interventions and dividing it into paragraphs, proves to be truly challenging since there are no tools that integrate all these requirements simultaneously.

This is the reason why the need to develop a tool that combines speaker diarization, audio-to-text transcription, and paragraph segmentation was identified. To expedite, enhance, and perform these processes, there are currently pre-trained Artificial Intelligence models available that can fulfill these functions, which means that these three models must be integrated.

For speaker diarization, Pyannote.Audio has been used, which contains pre-trained models for speaker diarization. This model provides a list of each intervention with the speaker's identifier, as well as the exact milliseconds of the intervention's start and end. Another model used is Whisper, OpenAI's automatic transcription model that allows transcribing audio into various formats, including text. Lastly, GPT-3.5, also from OpenAI, is utilized. It is a self-regressive learning model designed to generate written text automatically. In this case, using the appropriate prompt enables the task of paragraph segmentation.

As a result of developing this tool, a bridge has been created between the Artificial Intelligence tasks of diarization, transcription, and paragraph segmentation of audio, facilitating the use of these Artificial Intelligence models for both professional and personal users.

The source code is available in GitHub, just for any user so that they can use this tool and upgrade it or modify it so it can be adjusted to the user needs.

Keywords: Automatic; Artificial Intelligence; Audio Transcription; Speaker Diarization; Paragraph Segmentation

Índice general

Índice de figuras

Índice de tablas

Índice de listados

1. Introducción

En la era digital en la que nos encontramos, la transcripción de audios a texto está demostrando ser una de las herramientas más valiosas en diversos campos, como pueden ser la transcripción de entrevistas y conferencias o la generación automática de subtítulos para videos. Sin embargo, la tarea de transcribir audios de manera precisa y rápida puede resultar desafiante debido a factores como la presencia de múltiples interlocutores o la necesidad de dividir el texto resultante en párrafos para su mejor comprensión.

Por esta razón, y por la necesidad de la época en la que vivimos, en la que prima la facilidad de búsqueda y de uso, donde existen todo tipo de aplicaciones, se vio la necesidad de desarrollar una aplicación web que permita unir la diarización de interlocutores, la propia transcripción del audio a texto y la segmentación en párrafos del texto resultante en un solo servicio, basándose en modelos de Inteligencia Artificial previamente entrenados, acercando y facilitando el uso de la Inteligencia Artificial a aquellos usuarios que se encuentran menos familiarizados con ella.

1.1. Objetivos

Este Proyecto de Fin de Grado (PFG) pretende desarrollar una aplicación web que suponga para los usuarios una herramienta visual que integre varias tecnologías de Inteligencia Artificial en torno a la transcripción de audio, como son la diarización de interlocutores, la transcripción de audios o la segmentación del texto resultante en párrafos. A fin de lograr esto, se han fijado los siguientes objetivos:

- Desarrollo de una aplicación web en la que el usuario pueda decidir si ejecutar whisper desde un servidor propio o desde la API de Whisper.
- Desarrollar una aplicación web que pueda realizar todo este proceso de diarización, transcripción y segmentación en párrafos con todo tipo de archivos de audio como .mp3, .m4a, .WAV, etc.

- Correcta diarización de las intervenciones de los interlocutores que participen en el audio, en el menor tiempo posible.
- Transcribir un audio a texto todo lo rápido que se pueda.
- Segmentar el texto resultante de la transcripción de tal manera que resulte más natural para el lector.

1.2. Motivación

En la actualidad existen un sinfín de herramientas de transcripción de audio a texto, sin embargo, estas herramientas devuelven un texto sin formato, es decir, sin saltos de línea que lo segmenten en párrafos, lo cual dificulta enormemente su comprensión. Si a esto le añadimos que en la mayoría de audios que se transcriben, existe más de un interlocutor y que ninguna de estas herramientas permite diferenciar qué fragmentos de texto pertenecen a intervenciones de un interlocutor u otro, resultando en un texto completamente inútil ya que no se puede comprender el significado de este.

Por ejemplo, muchas veces se graban reuniones o clases para poder repasarlas más tarde, sin embargo, normalmente estas son de una duración considerablemente grande, aproximadamente entre 1 y 2 horas dependiendo del temario del audio, lo cual suele ser muy difícil de leer en un texto sin formato alguno. Sin embargo, si conseguimos que cada interlocutor sea identificado y que el texto sea dividido en las intervenciones de los interlocutores identificados, y estas a su vez se dividen en párrafos atendiendo a una lógica de un párrafo por tema, la comprensión del usuario del texto resultante por parte del usuario aumentará considerablemente.

Esto se debe a que, segmentando un texto en párrafos no muy largos, de unas 40-70 palabras por párrafo, los textos resultan más fáciles de leer y ayudan a asimilar textos, detectando fácilmente las ideas generales del texto ya que estas se presentan de manera ordenada y lógica [1].

Un párrafo se encarga de marcar los diversos puntos de que consta un tema o de distinguir opiniones a favor y en contra [2]. Es por ello que, si se usa acertadamente, la segmentación en párrafos facilita en gran medida el trabajo de comprensión, pero de lo contrario, puede afectar a la comprensión del texto [3].

2. Marco Teórico

En este capítulo se expondrán todos los conceptos teóricos necesarios para la consecución del proyecto, con el objetivo de proporcionar al lector el marco teórico para entender el desarrollo del proyecto.

2.1. Diarización de Interlocutores

La diarización de interlocutores es la tarea de particionar un audio en fragmentos de tiempo en base a la identidad del interlocutor. Esto se lleva a cabo juntando una serie de bloques que, cada uno, se encarga de una tarea específica, como detectar la actividad de voz, cambio de interlocutor, clustering, etc.

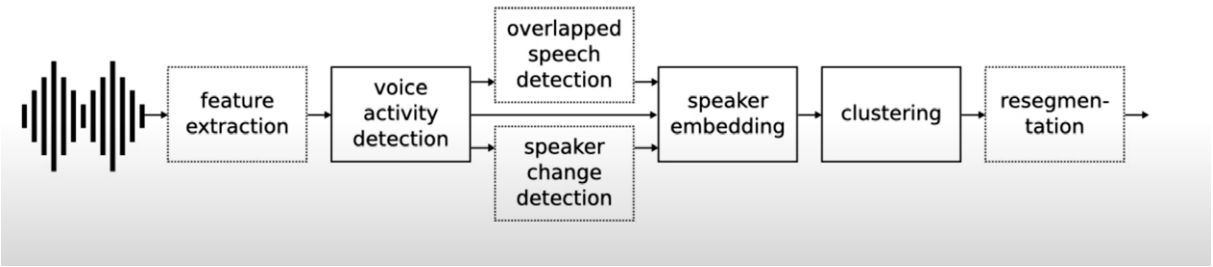


Figura 2.1. Flujo del pipeline de Pyannote.Audio (fuente: YouTube/HervéBredin)

Estos bloques, aunque deben ser entrenados por separado, Pyannote.Audio los combina en un pipeline de diarización, cuyos hiper parámetros son optimizados de manera conjunta para reducir el error y mejorar los resultados.

Estas son las tareas más importantes del pipeline de diarización de Pyannote.Audio:

- ***Speech activity detection***: es la tarea de detectar regiones en el discurso en un audio o grabación. Se trata de una tarea de etiquetado, donde el input es un conjunto de vectores de

características del audio y la salida esperada es una secuencia de 0 y 1 que significan que no hay nadie hablando o que se está hablando respectivamente.

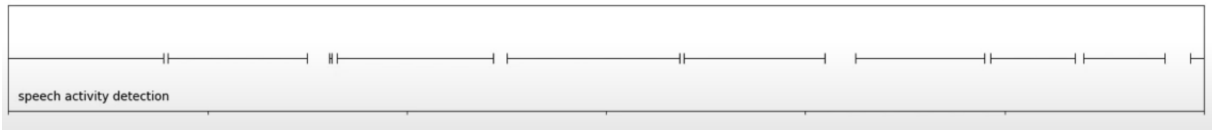


Figura 2.2. Salida de Speech activity detection (fuente: YouTube/HervéBredin)

- ***Speaker change detection***: se trata de detectar cambios de interlocutor durante el transcurrir de un audio/grabación. Esta etapa es similar a la anterior, sigue el principio de etiquetado, salvo que ahora los 1 significan cambio de interlocutor y los 0 son que el interlocutor no varía.

Figura 2.3. Salida de Speaker change detection (fuente: YouTube/HervéBredin)

- ***Overlapped speech detection***: se encarga de identificar los puntos en los que regiones en las que, al menos 2 interlocutores están hablando simultáneamente. Como las 2 anteriores sus resultados siguen un principio de etiquetado de unos y ceros. En este caso la salida esperada es de 0 si hay de 0-1 interlocutores hablando y 1 si hay 2 o más hablando simultáneamente.

Figura 2.4. Salida de Overlaped Speech detection (fuente: YouTube/HervéBredin)

- ***Speaker embedding***: la mayoría de los modelos más efectivos de clustering para la diarización de interlocutores requieren que este proceso se realice antes de llevar a cabo la clusterización. Pyannote.Audio implementa una gran cantidad de técnicas de métricas de aprendizaje como la pérdida de margen angular o la pérdida contrastiva.
- ***Clustering***: en toda diarización de interlocutores la etapa de clustering es la más importante de todas. Básicamente consiste en agrupar segmentos de audio de acuerdo a la identidad del interlocutor de esos segmentos [4].

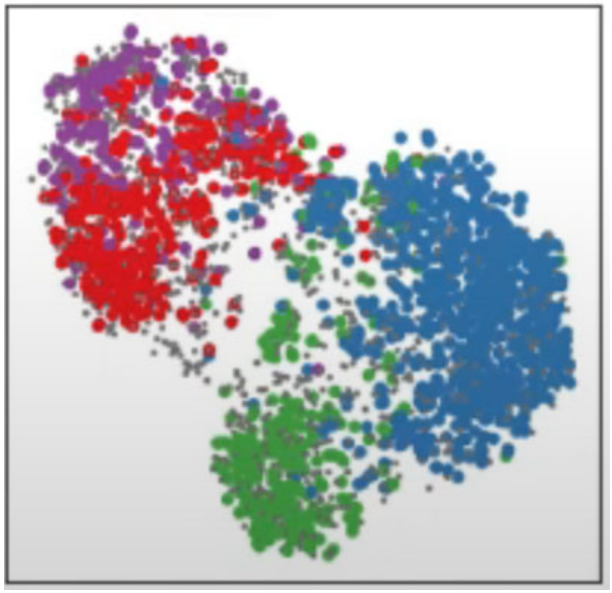


Figura 2.5. Salida de un Clustering (fuente: YouTube/HervéBredin)

Además, para que la diarización sea más fiable este proceso se divide en tres etapas principales: primero existe una segmentación local de hablantes, que se aplica a ventanas de tiempo cortas de unos 5 segundos con una superposición de unos 500 milisegundos, para a continuación pasar a una representación local neuronal de cada hablante. Por último, se lleva a cabo un agrupamiento jerárquico y global, es decir, que, en vez de ventanas de tiempo cortas, se aplica a ventanas de tiempo más grandes (unos 30 segundos) y sin superposición. Esta segmentación de la diarización facilita la resolución de toda la tarea [5].

Otro aspecto a tener en cuenta es que pese a que Pyannote te da la posibilidad de usar un modelo creado por ti mismo, también ofrece una serie de modelos de diarización ya entrenados que tienen una alta efectividad [6].

El resultado de todo este pipeline de diarización de interlocutores de Pyannote.audio es una lista con todas las intervenciones identificadas. Cada intervención de esta lista resultante incluye el identificador con el que se identifica el interlocutor en esa intervención, y también el milisegundo del audio en el que inicia dicha intervención y también el milisegundo en el que esta finaliza.

```python
from pyannote.audio import Pipeline

pipeline = Pipeline.from_pretrained("pyannote/speaker-diarization",
                                    use_auth_token="token_usuario")
diarization = pipeline(file_path)

for turn, _, speaker in diarization.itertracks(yield_label=True):
    print(f"[{speaker}] --> ({turn.start}, {turn.end})")
```

Output:

[Speaker_00] → (497, 7922)

[Speaker_02] → (7922, 10217)

[Speaker_00] → (10217, 13947)

[Speaker_00] → (13947, 20460)

Figura 2.6. Ejemplo de diarización

2.2. Whisper

Whisper [7], es un ASR (Automatic speech recognition) open-source entrenado durante 680000 horas en un entrenamiento supervisado con datos multilenguaje y multitarea obtenida de la web, lo cual lo hace más robusto, haciendo que los acentos, el sonido de fondo o un lenguaje técnico no sea un problema.

El input de Whisper es un audio, que posteriormente es dividido en fragmentos de 30 segundos, que se convierte en un Espectrograma de Mel (es la representación del sonido en forma de ondas, en el que el eje y se basa en los Coeficientes Cepstrales en las Frecuencias de Mel o MFCC [8]).

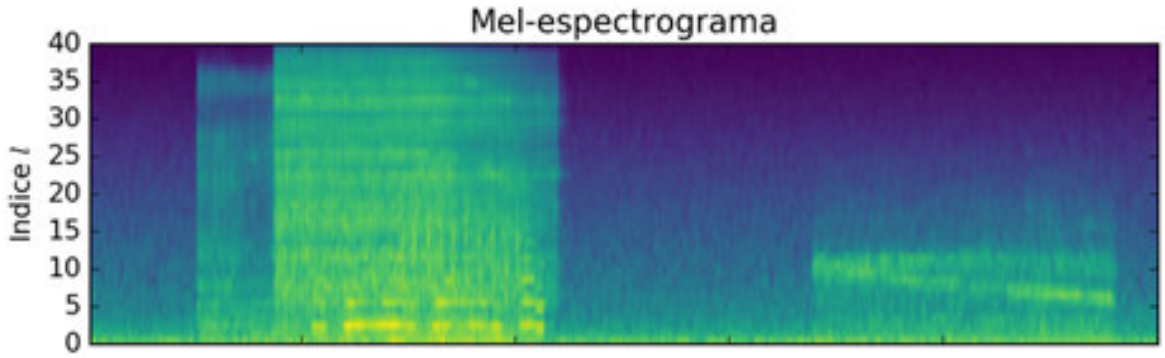


Figura 2.7. Espectograma de mel (fuente: OpenAI/Whisper)

Después el fragmento se le pasa a un codificador. A continuación, se entrena un decodificador para predecir el texto resultante de dicho fragmento, mezclado con tokens especiales que sirven para llevar a cabo tareas como la identificación del lenguaje, traducir al inglés. etc [7].

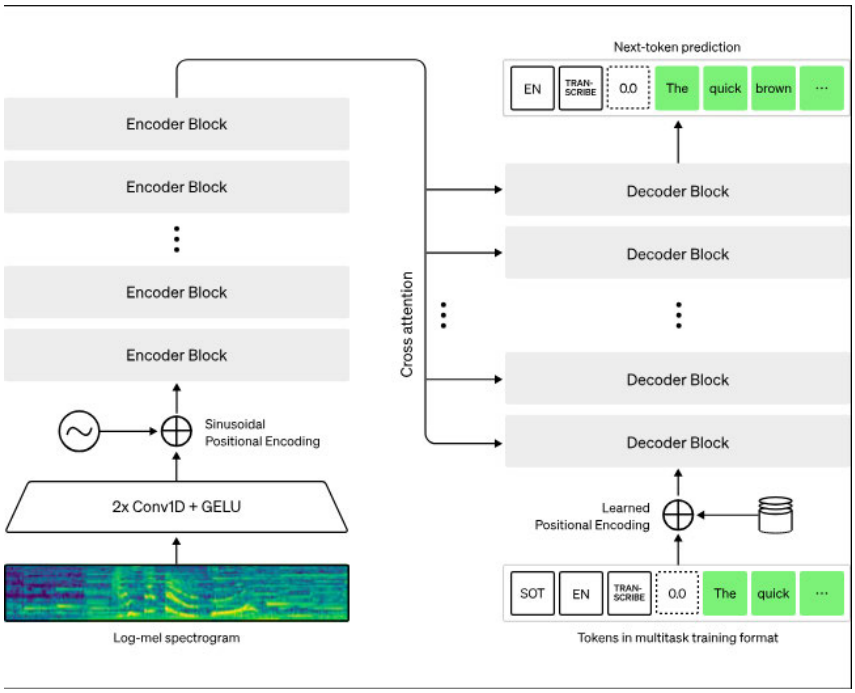


Figura 2.8. Flujo de Whisper (fuente: OpenAI/Whisper)

Además, Whisper ofrece distintos modelos ya entrenados, que van de modelos más pequeños, que por tanto son menos precisos pero rápidos de ejecutar requiriendo menos memoria, hasta modelos muy grandes, lo cual los hace muy pesados requiriendo mucha memoria, pero su precisión es mucho mayor [9]. Sin embargo, el seleccionar el modelo a ejecutar solo está disponible al ejecutar Whisper en local, es decir, tras haberlo instalado en un servidor propio. Sin embargo, la API de Whisper se ejecuta siempre el mismo modelo, large-v2, que solo está disponible para la API de Whisper [10].

Size	Parameters	English-only model	Multilingual model	Required VRAM	Relative speed
tiny	39 M	`tiny.en`	`tiny`	~1 GB	~32x
base	74 M	`base.en`	`base`	~1 GB	~16x
small	244 M	`small.en`	`small`	~2 GB	~6x
medium	769 M	`medium.en`	`medium`	~5 GB	~2x
large	1550 M	N/A	`large`	~10 GB	1x

Figura 2.9. Modelos disponibles para Whisper en local (fuente: GitHub/Whisper/Languages & Models)

Otro factor que afecta claramente al rendimiento de Whisper, lógicamente, es el idioma del audio. Whisper está disponible para un total de 57 idiomas diferentes, siendo el español el idioma más preciso, seguido por el italiano y el inglés [9].

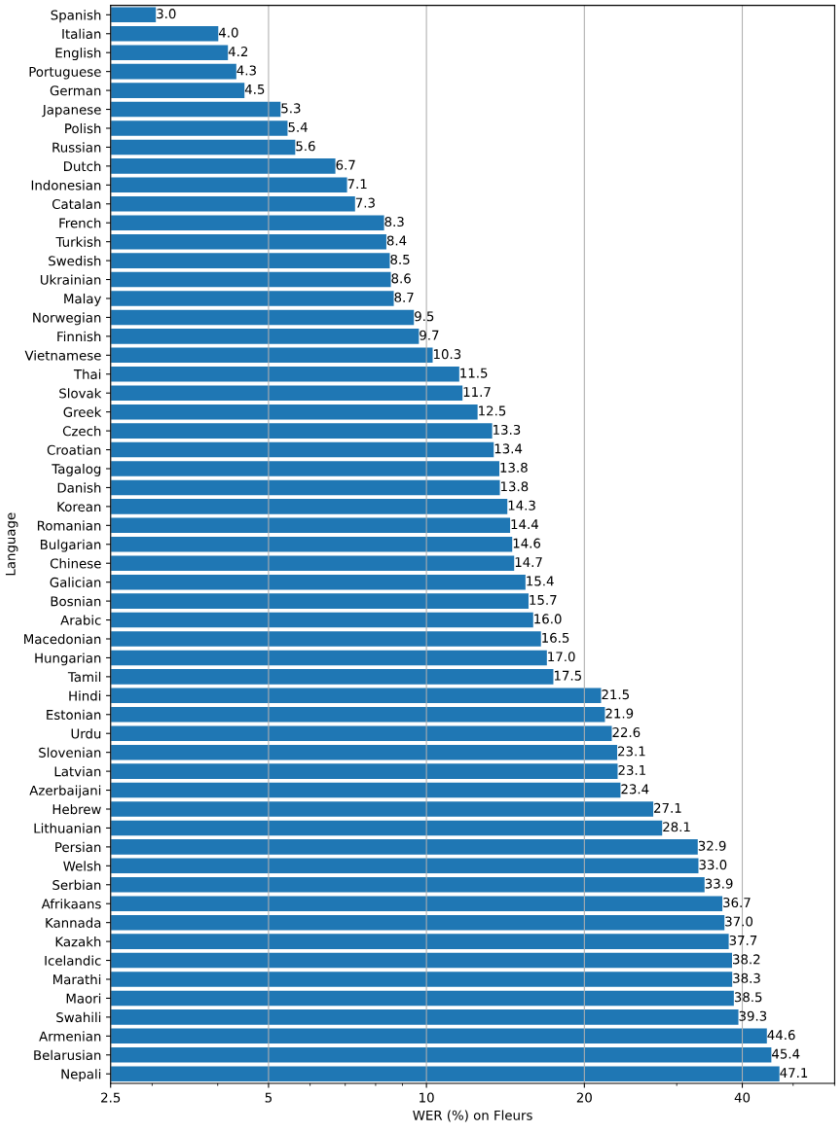


Figura 2.10. Idiomas disponibles para Whisper en local (fuente: GitHub/Whisper/Languages & Models)

Como resultado de todo este proceso se obtiene el texto resultante de la transcripción del audio, pero sin ningún formato, es decir, sin párrafos o saltos de línea que identifiquen los interlocutores de la conversación.

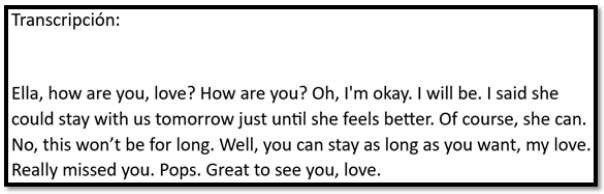


Figura 2.11. Ejemplo salida Whisper

2.3. GPT-3.5

GPT-3.5 (Generative Pre-trained Transformer 3.5) es una mejora de la tercera versión del modelo de lenguaje de OpenAI. Este es un modelo generativo, es decir, que puede generar largas cadenas de texto único como output [11]. Este modelo de lenguaje ha sido entrenado con 175 billones de parámetros que son optimizados durante la fase de entrenamiento [12].

GPT-3.5 es un modelo de Machine Learning de redes neuronales que puede, con un texto dado como entrada, transformar dicho texto en lo que el modelo predice que será el resultado más verosímil. Esto se ha logrado recopilando una gran cantidad de texto de internet, en torno de unos 570GB para encontrar patrones en un proceso llamado pre-entrenamiento afinado y generativo. Esto le permite realizar una gran cantidad de tareas de lenguaje, como puede ser traducir, completar textos o responder a preguntas [13].

Este modelo usa RLHF, el cual es un subcampo de la Inteligencia Artificial, que se centra en usar el feedback de un ser humano para mejorar los resultados de un algoritmo de Machine Learning. Al fin y al cabo, el objetivo de RLHF es incorporar el conocimiento y la experiencia humana a los algoritmos de Machine Learning.

Es por eso que durante la fase de entrenamiento un equipo de supervisión da feedback sobre el rendimiento del algoritmo del modelo y se producen una serie de reajustes, repitiéndose el proceso. Este feedback puede aportarse de diversas formas: "recompensado" o "castigando" las acciones del modelo, etiquetar los datos que quedan sin etiquetar o ajustar los parámetros del modelo [14].

Además, este modelo es entrenado con un rango amplio de lenguajes y de tareas y posee una arquitectura basada en una red neuronal de Transformers. Este Transformer consiste en una serie de capas de codificadores y decodificadores, que se entrenan con una gran cantidad de datos para entender los patrones del lenguaje. Todo esto permite al modelo poseer un amplio entendimiento de la complejidad de la comunicación humana [13].

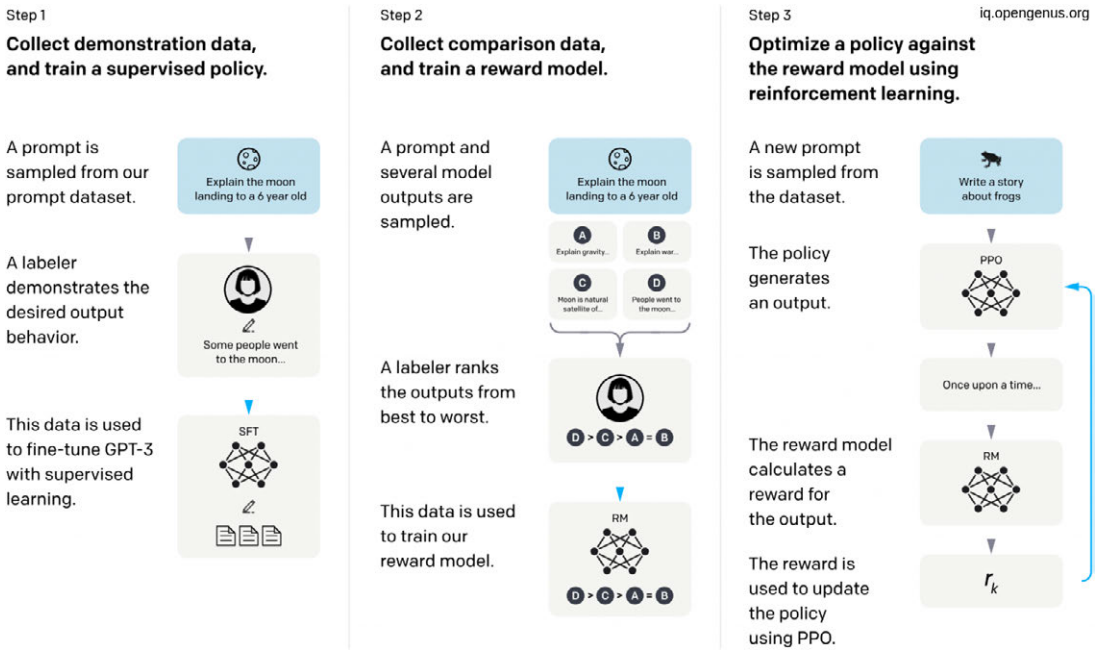


Figura 2.12. Ejemplo de entrenamiento GPT-3.5 (fuente: opegenus/gpt-3-5-model)

Con todo esto lo que se consigue es que GPT-3.5 procese un texto como input y así procese una gran variedad de tareas de lenguaje natural como puede ser:

- Generar textos no ficticios, como pueden ser diálogos, ensayos, imitaciones, nuevos artículos, etc.
- Generar textos profesionales, como emails, CV, anuncios, etc.
- Código de Python, SQL, javascript, CSS, HTML, etc.
- Generar textos creativos, como de ficción, poesías. Canciones, humor, etc.
- Habilidades racionales, maneja la lógica, el sentido común, las analogías, etc.
- Cuestiones filosóficas, como el sentido de la vida y otras cuestiones filosóficas [12].

Sin embargo, este modelo tiene una limitación en cuanto a la extensión del prompt (la pregunta que se le hace a GPT3) y la respuesta que proporciona este no pueden superar, en conjunto, un máximo de 4096 tokens para asegurar así que la respuesta se de calidad. Esta condición fue así durante todo el desarrollo de la herramienta. Sin embargo, en los últimos días, el 13 de junio de 2023, se lanzó GPT-3.5-16K, el cual tiene un máximo de 16000 tokens [15].

Así pues, los modelos referidos como GPT-3.5 son una serie de modelos entrenados antes del primer cuarto de 2021. Estos son los siguientes modelos de la serie de GPT-3.5:

- code-davinci-002, es el modelo base muy bueno para tareas de creación de código.
- text-davinci-002, es un modelo InstructGPT basado en code-davinci-002. InstructGPT, es GPT, pero añadiéndole RLHF.
- text-davinci-003, el cual es una mejora de text-davinci-002 [14].

2.4. Streamlit

Streamlit es una librería de Python que permite transformar en cuestión de minutos scripts de Python en aplicaciones web que pueden compartirse fácilmente. Está completamente escrita en Python, es open-source y gratuita [16].

Gracias a que está desarrollada en Python resulta una herramienta ideal para desarrollar aplicaciones web para proyectos de Data Science y Machine Learning en un corto periodo de tiempo ya que es compatible con librerías como scikit-learn, Keras, PyTorch, etc.

Una de las ventajas de Streamlit es que no necesita callbacks, pues todos los widgets son tratados como variables. Además, Streamlit posee una gran velocidad de ejecución de pipelines, pues implementa el almacenamiento de datos en caché, lo cual simplifica y acelera considerablemente dichos pipelines.

Una de las opciones que implementa Streamlit es la de revisar y cambiar en caso de que se produzca alguna modificación en el repositorio de Git vinculado. Estos cambios además ocasionan que la aplicación se despliegue automáticamente con los cambios introducidos, en el link compartido [17].

Es por todo esto que Streamlit se considera una herramienta ideal para desarrollar aplicaciones web usando Python, y sobre todo para aquellos proyectos que incluyan tareas de Data Science y de Machine Learning.

2.5. Metodología basada en prototipos evolutivos

La metodología basada en prototipos es una metodología evolutiva, pues se empieza con un prototipo pequeño y este va evolucionando, enfocándose en diseñar, implementar, medir y ajustar un plan hasta resultar en el producto final. El prototipo siempre tiene que ser evolutivo, gracias a la interacción con los usuarios y funcional, ya que se trata de una app que funciona [18].

Todo esto permite que el sistema, o parte de este se construya rápidamente y comprender y aclarar ciertos aspectos en los que se aseguren que desarrollador, cliente y usuario están de acuerdo en lo que necesita, minimizando el riesgo y la incertidumbre del proyecto.

Esta metodología se suele aplicar cuando el cliente define unos objetivos generales, sin delimitar la entrada ni la salida, es decir, cuando no está seguro de la eficacia de algún algoritmo, de la adaptabilidad del sistema. etc [19].

Esta metodología es iterativa y se compone de las siguientes fases:

- ***Fase 1, Extracción de requisitos y análisis***: En esta fase se identifican y definen con detalle los requisitos del sistema. Durante este proceso se entrevistan a los usuarios del sistema y al cliente para conocer sus expectativas sobre el sistema.
- ***Fase 2, Diseño rápido***: se crea un diseño simple del sistema a desarrollar. Sin embargo, este diseño no es uno completo, más bien sirve para que el usuario y el cliente puedan hacerse una idea sobre el sistema y también los desarrolladores, ayudándolos a desarrollar el prototipo.
- ***Fase 3, Construir un prototipo***: basándose en la información obtenida de las dos fases anteriores, que son los requisitos identificados y el diseño creado, se desarrolla un prototipo. Este prototipo representa un pequeño modelo del sistema requerido el cual funciona.
- ***Fase 4, Evaluación inicial del usuario***: se presenta el sistema propuesto al cliente para la evaluación inicial. Esta fase es de gran utilidad, ya que ayuda a encontrar los puntos fuertes y débiles del modelo funcional, gracias a que el cliente y los usuarios realizan una serie de comentarios y sugerencias, las cuales son proporcionadas a los desarrolladores.
- ***Fase 5, Refinar el prototipo***: Si el usuario no está contento con el prototipo presentado en la fase anterior, el prototipo debe de ser refinado de acuerdo al feedback y las sugerencias del cliente y de los usuarios. Esta fase no finalizará hasta que los requisitos especificados en la fase anterior se cumplan. Una vez se cumplan se vuelve a presentar al cliente y usuarios. Si

tienen algún comentario a sugerencia se vuelve a refinar el prototipo. Por el contrario, si el prototipo es aprobado, se ha desarrollado entonces el sistema final.

- ***Fase 6, Implementar el producto y mantener***: Una vez el sistema final ha sido desarrollado, se debe testear y desplegar a producción. Además, el sistema sufre mantenimiento rutinario para minimizar el tiempo de caída y prevenir fallos de gran escala [20].

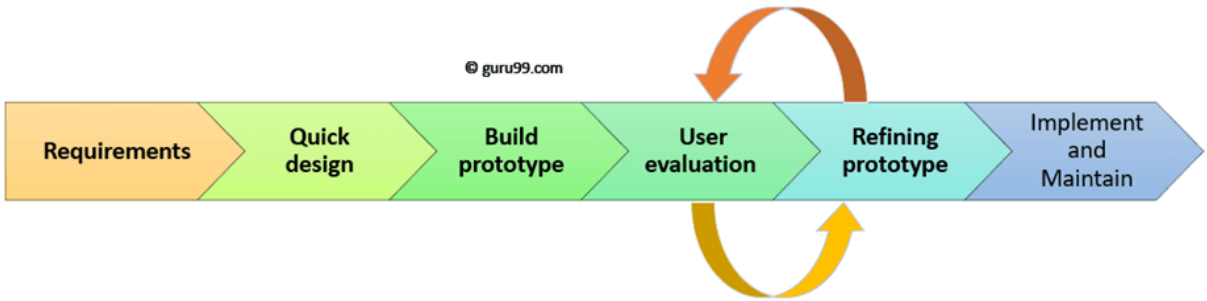


Figura 2.13. Ciclo de metodología con prototipado (fuente: guru99/software-engineering-prototyping-model)

3. Desarrollo del proyecto

3.1. Análisis de requisitos y especificaciones

El objetivo principal del proyecto es la creación de un texto que contenga la transcripción correspondiente a un audio dado. Este texto puede tener distintos formatos dependiendo de las opciones que seleccione el usuario para la transcripción del audio, las cuales pueden ser:

- Diarización de interlocutores activada o desactivada.
- Segmentación en párrafos activada o desactivada.
- Traducción del audio a inglés activada o desactivada.
- Ejecutar Whisper en un servidor propio o a través de su API. En caso de seleccionar la API se añade otro parámetro que es la clave de la API de OpenAI, y si selecciona servidor propio se elimina el parámetro de la clave por el modelo con el que ejecutar Whisper.

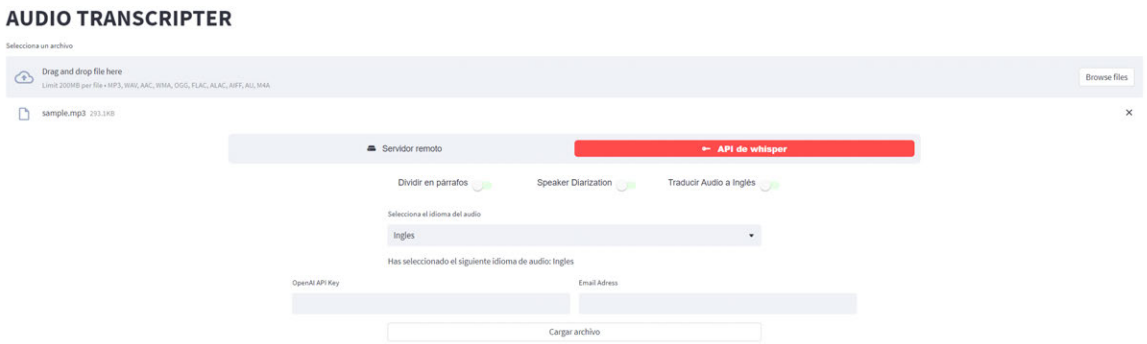


Figura 3.1. Parámetros herramienta API de Whisper

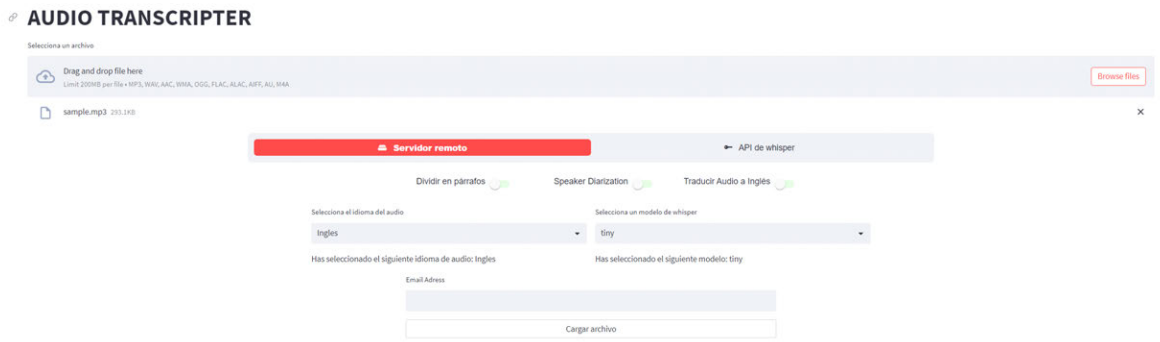


Figura 3.2. Parámetros herramienta Servidor propio

En función de estos parámetros se creará un texto en un formato u otro. Por defecto, se lleva a cabo una transcripción sin diarización de hablantes, traducción o segmentación en párrafos. Otros parámetros necesarios para llevar a cabo la transcripción son: el correo al que enviar el texto resultante, el idioma del audio y el fichero de audio que se quiere transcribir, el cual puede estar en cualquier formato, pues uno de los requisitos es que se permita la transcripción de todo tipo de fichero de audio, ya sea .mp3, .WAV, .m4a, etc.

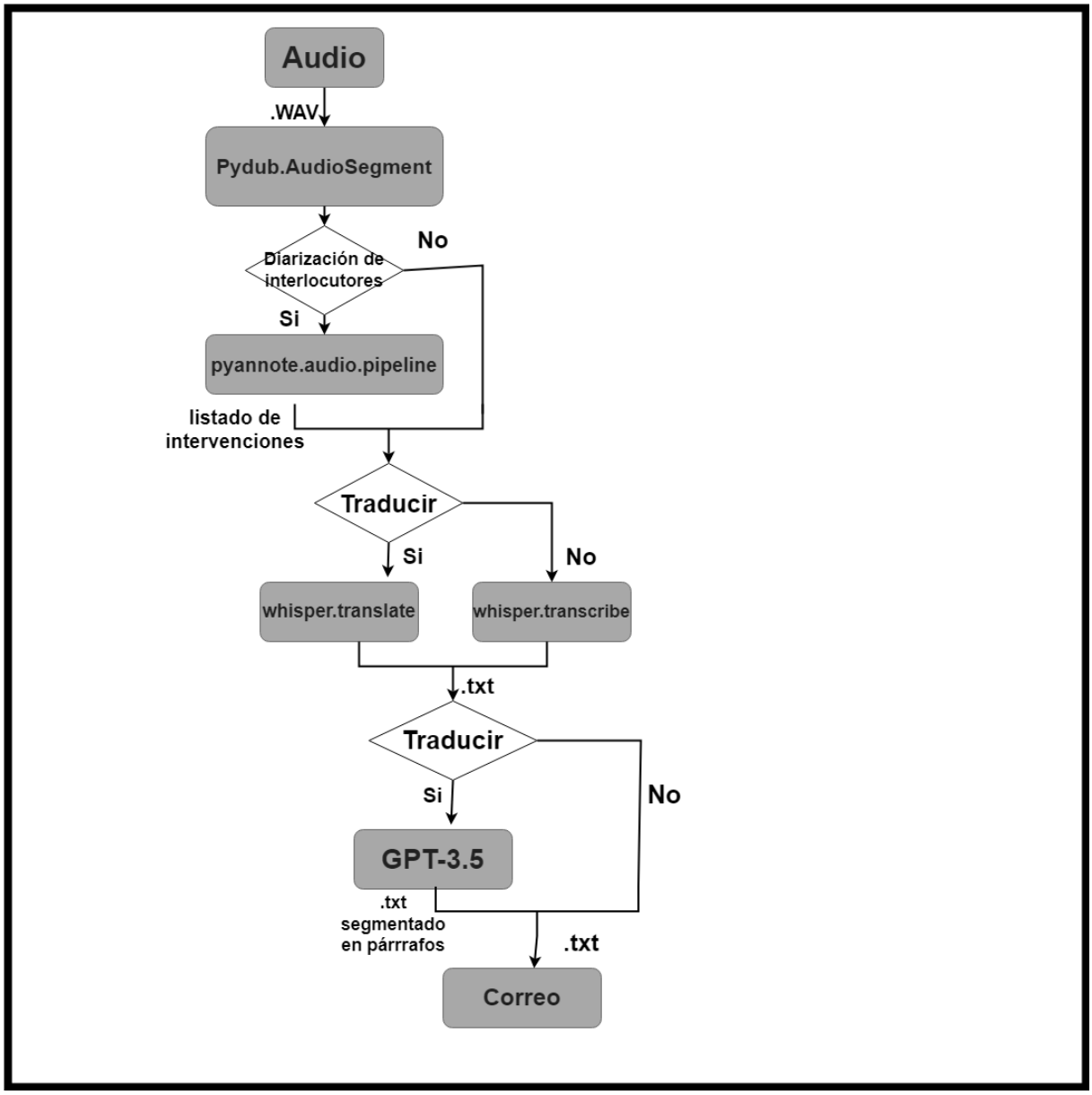


Figura 3.3. Diagrama de funcionamiento de la herramienta

La figura 3.3 representa, a alto nivel, el funcionamiento de la herramienta. Empieza con la fuente de información que es el audio a transcribir. Este puede estar en cualquier formato, pues una vez seleccionado el audio este es tomado por pydub.AudioSegment que exporta el contenido del fichero de audio, sea cual sea su extensión, a un archivo .WAV. Posteriormente si el usuario ha seleccionado la opción de diarización el archivo .WAV será tomado por *Pyannote.Audio.Pipeline* para aplicar la diarización de interlocutores pasando una lista de los fragmentos de las intervenciones sobre los que se aplicarán las siguientes fases. A continuación, encontramos la siguiente casuística:

- Si no ha seleccionado la diarización se aplicarán el resto de fases al archivo de audio al completo y no a cada intervención.
- A continuación, el fichero de audio y, si se ha realizado la diarización de interlocutores, opcionalmente una lista de las intervenciones es tomada por Whisper, que con estos parámetros que le hemos pasado, lleva a cabo la transcripción o la traducción del audio, dependiendo de la opción seleccionada por el usuario. Sin embargo, ambos métodos retornan un texto el cual se corresponde con el contenido del audio ya sea en un idioma u otro.
- Por último, si el usuario ha seleccionado la segmentación en párrafos, el texto resultante anterior es tomado por el modelo GPT-3.5 que, junto con un determinado prompt es capaz de segmentar el texto en párrafos. En este prompt se le solicita al modelo que divida el texto en párrafos no muy extensos de tal manera que se cumpla 1 tema por párrafo. Además de esto, se le dan una serie de limitaciones, como que no altere ninguna palabra o que no cambie el idioma del texto. Para acabar, se le especifica el formato en el que debe mostrar el texto resultante.

Una vez realizados todos estos procesos, se muestra el texto resultante en la aplicación y se envía un correo a la dirección de correo electrónico introducida como parámetro.

3.1.1. Conversión del fichero de audio

Debido a que la aplicación debe ser capaz de ofrecer la diarización de hablantes en todos los formatos de audio y que Pyannote.Audio solo es capaz de realizar la diarización de interlocutores a archivos .WAV, uno de los requisitos principales es el de convertir el audio que el usuario seleccione en un .WAV independientemente de su formato de origen.

3.1.2. Tratamiento de la duración del audio

Uno de los requisitos a la hora de ejecutar Whisper a través de la API de OpenAI es que el audio en cuestión no tenga un tamaño mayor que 25MB. Si el archivo que el usuario quiere transcribir es más pesado que esto el audio debe dividirse y transcribirse en dos fragmentos de audio diferente. Este proceso es muy delicado, ya que, en caso de que la calidad del audio no sea muy buena, como, por ejemplo, un audio con mucho ruido de fondo, Whisper suele detectar palabras al final del audio debido a que detecta ese ruido de fondo como palabras. Por ello en cada zona de fragmento se

trascribe un pequeño solapamiento de unos 30 segundos antes y después de que se produzca el corte. Este solapamiento se usa para comparar con el fragmento transcrito y cortar donde empieza el relleno.

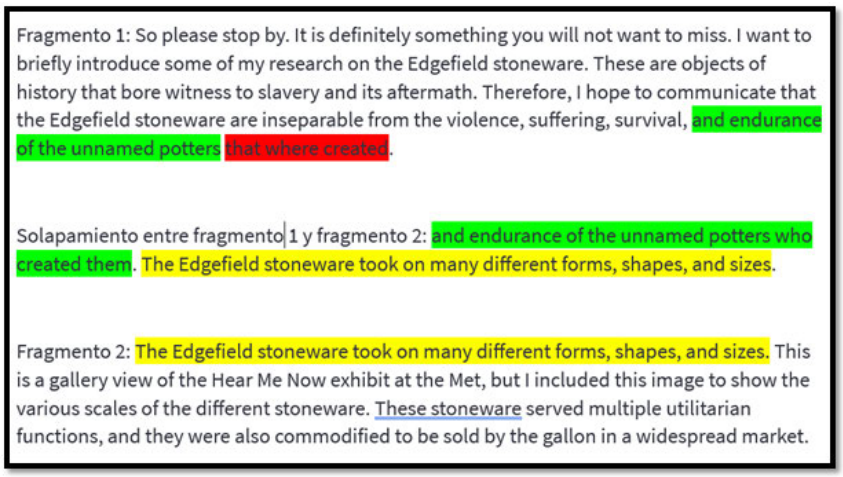


Figura 3.4. Ejemplo tratamiento de transcripción de audios de más de 25MB

En la figura 3.4 se puede ver un ejemplo de cómo se trata las transcripciones resultantes de los distintos fragmentos de un mismo audio de más de 25MB.

- La zona verde es la parte común entre el fragmento 1 y el solapamiento.
- La zona amarilla es la parte común entre el fragmento 2 y el solapamiento.
- La zona roja es la parte del fragmento 1 que Whisper se inventa durante la transcripción.

Sabiendo esto, se puede modificar el fragmento 1 para que la transcripción sea más precisa. Para ello se corta el fragmento al inicio de la parte verde y se cambia por la parte verde del solapamiento, sin añadir la parte amarilla, pues eso pertenece al siguiente fragmento.

3.1.3. Tratamiento del texto a segmentar en párrafos

Otro requisito sería la segmentación en párrafos del texto usando GPT3 a través de la API de OpenAI. Sin embargo, al igual que con Whisper, esta limita la extensión del texto que puede procesar en una sola llamada a dicha API. En concreto, la suma de la extensión del prompt y la respuesta de la API no puede superar un máximo de 4096 Tokens, lo cual para audios cortos no es un problema.

Sin embargo, con audios largos, el texto resultante de la transcripción puede llegar a ser realmente largo. La solución rápida parece segmentar el texto cortando por número de tokens, pero esto seguramente puede resultar en dividir una frase a la mitad, lo cual no es lo correcto. Para evitar esto se debe cortar el texto en fragmentos mirando el número de tokens, pero también ha de tener en cuenta de mantener las frases completas. Por ejemplo, si el fragmento de texto actual tiene 4000 tokens y la siguiente frase es de 100 tokens, esta no se añade ya que esta superaría el máximo número de tokens permitidos y habría dejar la frase incompleta.

3.1.4. Ejecución eficiente

Este proceso pude llegar a ser muy largo para audios de cierta extensión. Es por ello que uno de los requisitos es el de acelerar el proceso lo máximo posible. En este servicio los procesos que más tardan son la diarización de hablantes y la transcripción del audio.

Para lograr acelerar la diarización de hablantes pyannote.Audio posee un método que permite ejecutarlo en una GPU. Es por ello que el fichero .WAV se transferirá a un servidor propio, el cual posee varias GPUS, y se ejecutará en un contenedor de Docker [21], que cargará la lista resultante de la diarización en un archivo pickle que posteriormente se transferirá al servidor donde se aloja nuestra aplicación.

Para la trascripción del audio existen dos vías de ejecución. Una de ellas es la API de Whisper, la cual ya es bastante rápida de por sí y no se puede hacer nada para acelerar un poco más el proceso. La otra opción es ejecutar Whisper en local, la cual sí que se puede acelerar, pues al igual que Pyannote.Audio puede ser ejecutada en una GPU, por lo que el audio también será procesado en el mismo servidor propio que posee varias GPUs, obteniendo el texto correspondiente en el servidor que aloja nuestra app.

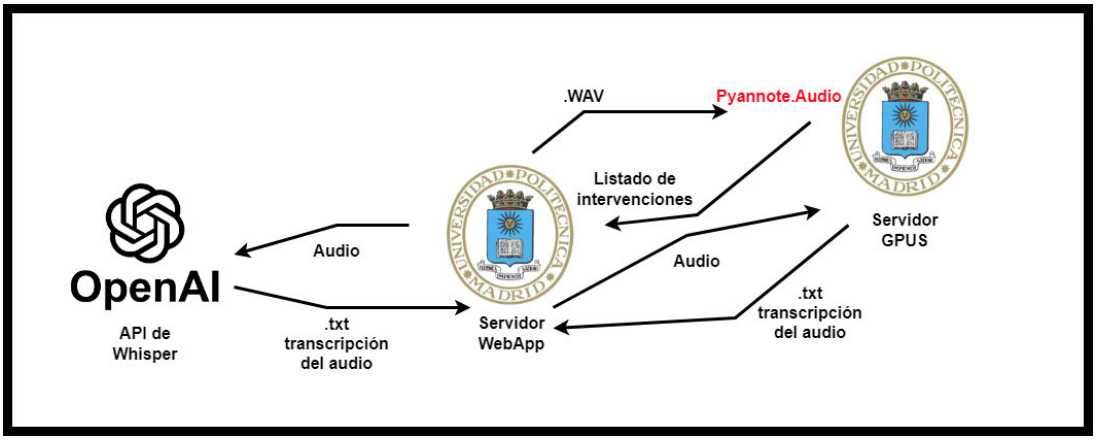


Figura 3.5. Diagrama de servidores

3.2. Desarrollo

El proceso de desarrollo del proyecto consta de las siguientes fases o etapas

3.2.1. Fase 1: Diarización de interlocutores

Esta es la primera fase, relativa a la detección de las intervenciones de cada interlocutor con Pyannote.Audio, y usando uno de los modelos pre-entrenados que tiene esta biblioteca. Esta fase solo se ejecutará en caso de que el usuario seleccione la opción de diarización de interlocutores. Primero se convierte el formato del audio a .WAV sea cual sea su extensión original, para después subir el audio al servidor propio y realizar la llamada a Pyannote.Audio. Una vez hecha la llamada al pipeline de Pyannote.Audio desde el servidor propio con varias GPUs, para acelerar su ejecución, se genera en este servidor un archivo pickle que se pasará al servidor donde se aloja la aplicación. Una vez abierto ese archivo pickle se obtendrá una lista que contiene el identificador del interlocutor, el tiempo de inicio y el de fin para cada una de las intervenciones que reconoce Pyannote.Audio. Si el usuario no selecciona la opción de la diarización se devuelve una lista con un único elemento, el cual no tiene un identificador de interlocutor y tiene tiempos de inicio y fin, que reflejan el inicio y el fin del audio.

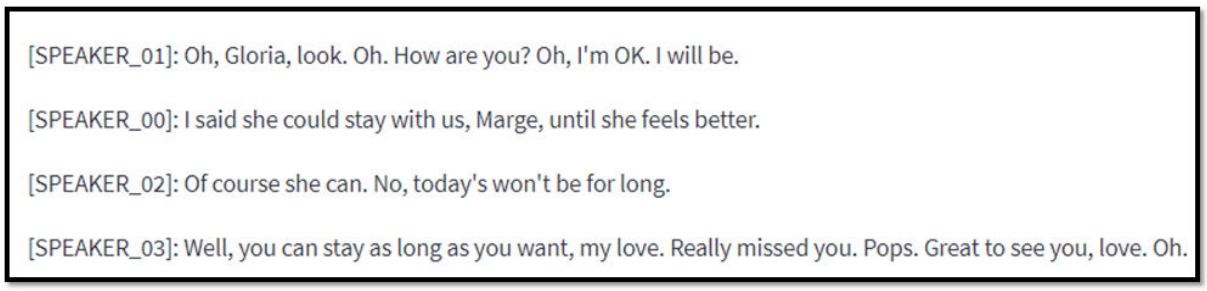
3.2.2. Fase 2: Transcripción del audio

En esta parte se procede a iterar sobre la lista resultante de la fase anterior, en el que, en cada iteración, del audio original se extrae el fragmento correspondiente a la intervención con los tiempos de inicio y fin de la lista y a este audio le aplicamos siempre el mismo proceso.

Este proceso consta en transcribir o traducir, en función de lo haya seleccionado el usuario, el fragmento de audio de cada iteración, ejecutando Whisper en un servidor propio o a través de la API de Whisper, también esto dependerá de lo que haya seleccionado el usuario. En caso de que se seleccione la opción del servidor propio primero se sube el fragmento de audio a dicho servidor propio y ejecutamos Whisper sobre ese audio, con las opciones que haya seleccionado el usuario, en dicho servidor propio obteniendo un archivo .txt cuyo contenido se almacenará en una variable.

Si, por el contrario, el usuario quiere transcribir el audio usando la API de Whisper, se deberá dividir el audio en fragmentos de 10 min. Si el audio original supera los 25MB, ya que esta API está limitada a audios no más grandes que 25MB, e ir haciendo llamadas a la API de whisper para cada uno de los fragmentos generados y se van concatenando las respuestas en una sola variable. Si no supera los 25MB se hace una sola llamada a dicha API pasándole el audio en su totalidad almacenando la respuesta de la API en una variable.

Al usar la Api de Whisper, cuando se divide el audio en fragmentos, si la calidad de este es baja o existe mucho ruido de fondo, Whisper puede llegar a añadir al final de la transcripción texto que no se corresponde con dicho audio, pues confunde el ruido con palabras sin sentido, por lo que debe transcribirse un fragmento que actúe como solapamiento de unos 30s en el momento de cada corte para comparar el solapamiento con la transcripción realizada y así comparar ambos 2 y eliminar la parte no común.


[SPEAKER_01]: Oh, Gloria, look. Oh. How are you? Oh, I'm OK. I will be.

[SPEAKER_00]: I said she could stay with us, Marge, until she feels better.

[SPEAKER_02]: Of course she can. No, today's won't be for long.

[SPEAKER_03]: Well, you can stay as long as you want, my love. Really missed you. Pops. Great to see you, love. Oh.

Figura 3.6. Ejemplo de trascripción de un audio tras la fase 2

3.2.3. Fase 3: Segmentación en párrafos

En esta tercera fase, si el usuario desea que el texto esté segmentado en párrafos, se procede a la segmentación del texto. Para ello se utiliza un modelo de aprendizaje generativo como es GPT-3.5, y para ello se utiliza la API de OpenAI, que, al igual que la de Whisper, está limitada en la longitud de la suma prompt y la respuesta, la cual no puede superar los 4096 tokens. Por ello se debe segmentar el texto en fragmentos de máximo 4096 caracteres, ya que 1 token equivale a aproximadamente 4 caracteres y así se deja caracteres libres para la respuesta, asegurándose que no se sobrepasa el límite de tokens.

La división en fragmentos no es solo siguiendo el criterio del número de caracteres si no también que no deje a mitad una frase. Por ello se va sumando frase a frase hasta que añadir una nueva frase más ocasione que el límite de caracteres sea superado.

Una vez dividido en fragmentos se procede a realizar una llamada la API de OpenAI para cada fragmento, la cual aparte del motor, que como mencionamos en apartados anteriores, es GPT-3.5 o "text-davinci-003", requiere otro parámetro que es el prompt, el cual es la tarea que tiene que realizar. Esta parte es muy importante ya que según que le pasemos la salida será una u otra, por lo que se debe ser muy específico con lo que se quiere que haga, cómo lo haga y como muestre el resultado. Por ello en el prompt le decimos lo siguiente:

```
pregunta = "Tengo un texto sin dividir en párrafos, es
decir, en una sola linea. Quiero que lo dividas en párrafos
sin cambiar ni una sola palabra. Un requisito importante es
que tenga un formato muy estricto, el cual es el siguiente:
[parrafo 1]: contenido párrafo 1\n[parrafo 2]: contenido
párrafo 2\n...[parrafo n]: contenido párrafo n. Una última
petición es que no escribas nada de introducción como por
ejemplo: Aquí tienes tu texto o algo similar. Aquí te
muestro el texto que quiero que dividas en párrafos tal y
como te he descrito: "
```

Figura 3.7. Promt para la segmentación del texto

Tal como se muestra en la figura 3.7, se le está explicando al modelo el contexto ("*Tengo un texto sin dividir en párrafos, es decir, en una sola linea.*") y diciendo que y como debe hacerlo ("*Quiero que me lo dividas en párrafos sin cambiar ni una sola palabra*") y como mostrar el resutlado ("*Un requisito importante es que tenga un formato muy estricto, el cual es el siguiente: [parrafo 1]: contenido párrafo 1\[parrafo 2]: contenido párrafo 2\...[parrafo n]: contenido párrafo n. Una última petición es que no escribas nada de*

introducción como por ejemplo: Aquí tienes tu texto o algo similar"). Es muy importante ser así de preciso pues GPT-3.5 es estocástico, lo que implica que para una misma pregunta puede retornar una respuesta diferente cada vez que se le formula. Por ello se debe de restringir al máximo la ambigüedad para que el resultado siempre sea el deseado.

Por último, una vez se han terminado todas las peticiones para los distintos fragmentos del texto se van concatenando en una variable que es el texto resultante final.

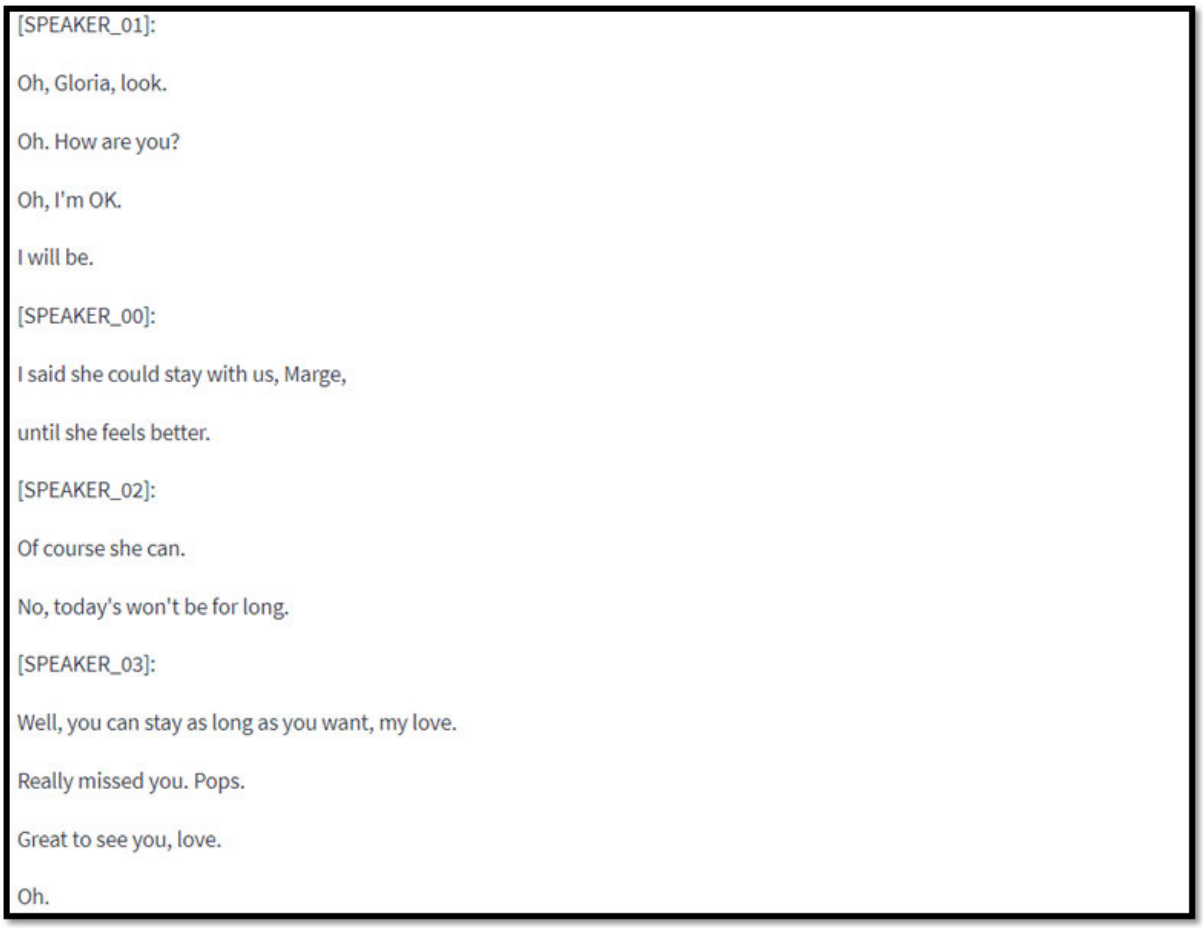


Figura 3.8. Ejemplo de transcripción de un audio tras la fase 3

3.2.4. Fase 4: Visualización del resultado

Por último, esta fase se encarga de la visualización de los resultados. Para ello, el texto obtenido en la fase 3 se muestra por pantalla, junto con un botón de descarga para descargar el texto que se muestra por pantalla en un fichero .txt, y si el usuario ha elegido la opción de segmentación en párrafos se le da al usuario la opción de ver el texto sin segmentar en párrafos.

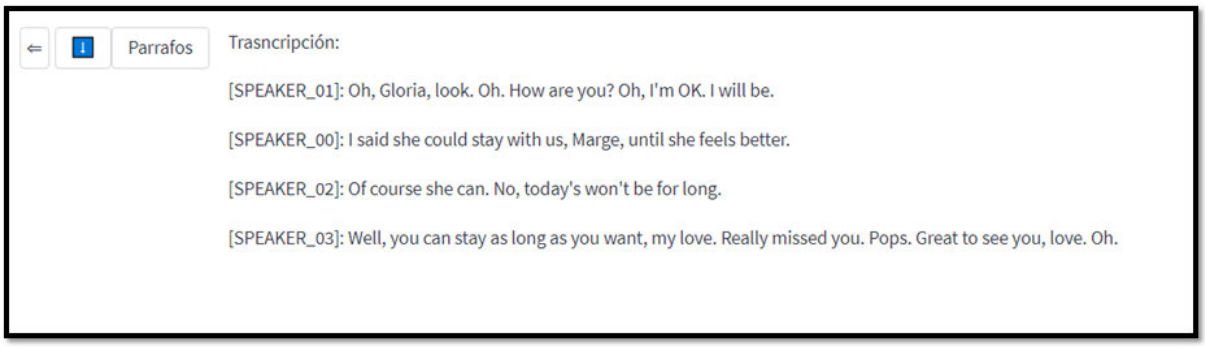


Figura 3.9. Ejemplo salida fase 4

Otra forma de visualizar el resultado es enviando un correo electrónico con el resultado. Esto se hace en dos pasos:

- Primero, antes de empezar con todo el proceso de ejecución, se comprueba que el correo introducido por el usuario es correcto.
- Luego se fórmula y envía el correo, de tal manera que el asunto del correo es el nombre del audio, el cuerpo es el texto resultante de toda la ejecución de la herramienta y el destinatario es la dirección introducida por el usuario.

3.3. Implementación

El código resultante de la implementación se encuentra en un repositorio de GitHub [22] para que cualquiera pueda utilizar y mejorar la herramienta o adaptarla a sus necesidades específicas.

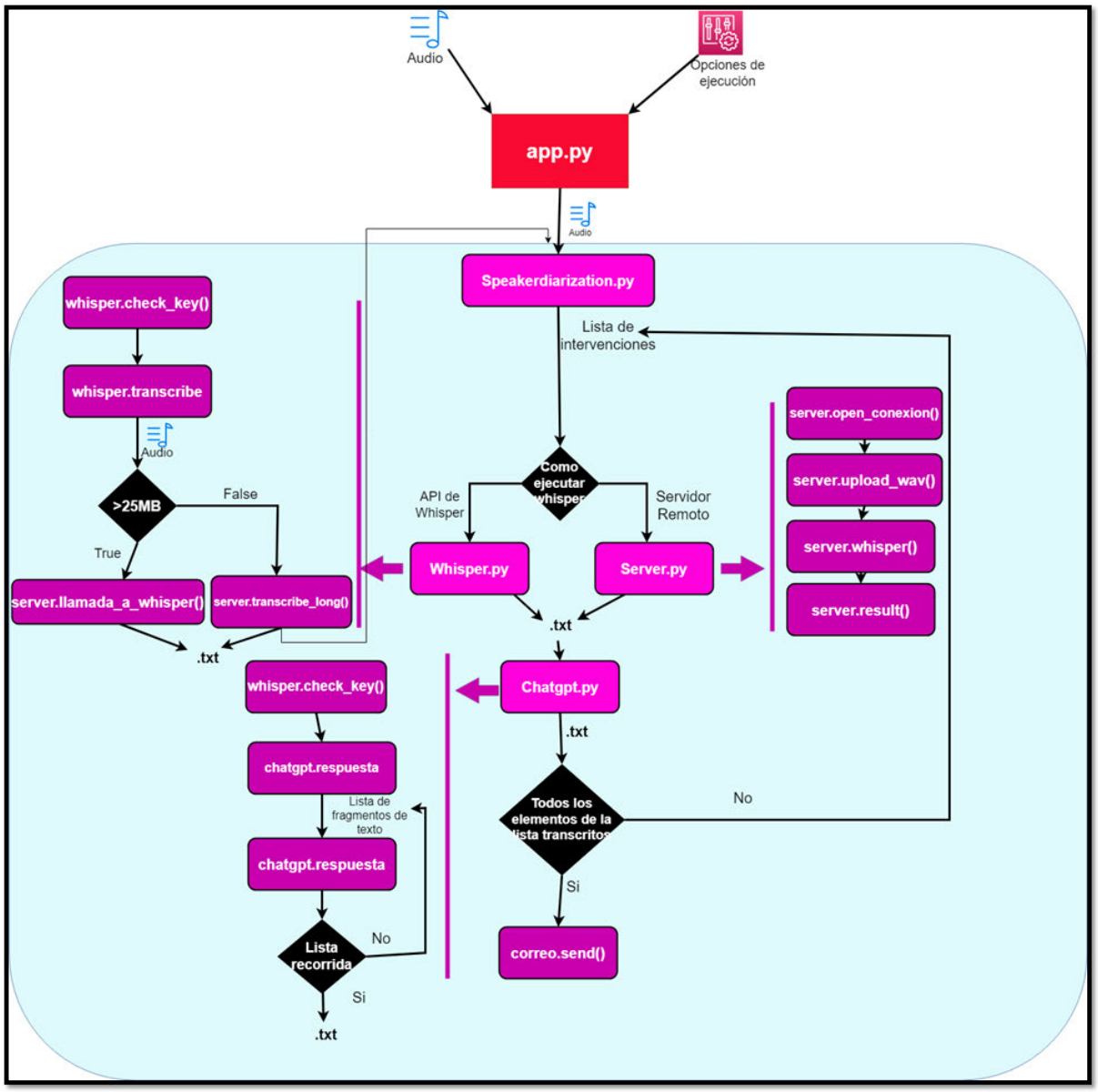


Figura 3.10. Diagrama de flujo del funcionamiento de app.py

En la figura 3.10 se representa el diagrama de flujo con todos los módulos y submódulos por los que pasa la ejecución del programa, los cuales van a ser descritos a continuación.

3.3.1. Diarización de interlocutores

La diarización de interlocutores es el módulo que se encarga del inicio de la ejecución del proyecto. La llamada a la ejecución de la diarización de hablantes se hace desde el fichero speaker_diarization.py

que a su vez es llamado por el fichero principal app.py, y se compone de los siguientes parámetros:

- *audio_file*: parámetro que establece el audio que se ha de diarizar como un stream de bytes.
- *barra*: parámetro que es un objeto de Streamlit para mostrar al usuario el progreso en la diarización.
- *progreso*: parámetro entero cuyo valor refleja el porcentaje que se ha completado en la ejecución.
- *diarizar*: parámetro booleano que refleja si el usuario desea o no diarizar los interlocutores. Si toma un valor False, entonces no hace nada y retorna una lista con un único valor.

Una vez cargado el fichero en el servidor propio, se ejecuta la diarización de hablantes en dicho servidor generando un archivo pickle que se envía al servidor donde se aloja la aplicación. Una vez abierto el archivo pickle en el código de la aplicación, se retorna la lista que contenía dicho archivo pickle. Cada elemento de esta lista toma tres valores: el identificador del interlocutor, el tiempo del audio en el que inicia y finaliza la intervención de ese interlocutor, ambos en milisegundos.

Si el usuario no ha seleccionado la opción de diarizar interlocutores, la herramienta no ejecuta la diarización y retorna una lista con un único valor formado por un identificador, y los tiempos de inicio y fin se corresponden con el inicio del audio y el fin del mismo.

Figura 3.11. Ejemplo de lista de la ejecución de speaker_diarization.py

3.3.2. Transcripción del audio.

En esta segunda parte del proyecto se itera sobre la lista resultante y se lleva a cabo la transcripción de cada fragmento, construyendo la transcripción resultante del audio. Como en este caso hay dos opciones para ejecutar Whisper, existen dos módulos principales: Server.py, el cual lleva a cabo la

transcripción del audio en un servidor propio, y Whisper.py, que realiza la transcripción del audio a través de la API de Whisper.

Server.py es la clase encargada de realizar tareas en el servidor propio, por tanto, tiene varios métodos para realizar acciones en el servidor propio, de las cuales estas son las que se usan para la transcripción del audio:

- *open_conexion()*: es el método encargado de abrir la conexión con el servidor propio. Este método no toma ningún parámetro.
- *upload(audio) y upload_wav(audio)*: estos dos métodos se encargan de subir al servidor propio el audio en stream de bytes que toman como parámetro, usando el método *open_conexion()*. La única diferencia es que el método *upload()* toma como parámetro la extensión del archivo a subir, mientras que *upload_wav()* llama a *upload()* con la extensión en .WAV.
- *idle()*: este método retorna la GPU que está más ociosa de las que posee el servidor propio. Este método no requiere ningún parámetro.
- *whisper(nombre_audio, lan, model, gpu, trad)*: es el método principal, se encarga de ejecutar Whisper en el servidor propio tomando como parámetros el nombre del fichero de audio subido por o bien *upload* u *upload_wav*, el lenguaje del audio, el modelo con el que ejecutar Whisper, la GPU del servidor sobre la que se quiere ejecutar Whisper y un booleano que indica si se quiere traducir o no.
- *result(nombre)*: este método retorna el contenido del archivo generado en el servidor propio. Toma como parámetro el nombre del fichero de texto generado por la ejecución de Whisper.

Whisper.py es la clase encargada de la transcripción del audio a través de la API de Whisper y, por tanto, también de llevar a cabo todo el tratamiento del audio necesario para que esté acorde a los requisitos de la API de Whisper, es decir que no pese el audio pasado a la API mas de 25MB. Estos son los métodos principales de la clase Whisper.py:

- *llamada_a_whisper(audio, lan, translate)*:este método toma como parámetros el audio a transcribir, el lenguaje del audio y un booleano que refleja si el usuario quiere traducir o no. Este método simplemente en función del valor del booleano realiza una llamada a *openai.Audio.transcribe(False)* o a *openai.Audio.translate(True)*. Retorna el texto resultante de la transcripción.

- *transcribe_long(audio, lan, translate)*: este método toma como parámetros el audio a transcribir, el lenguaje del audio y si el usuario quiere traducir o no. Básicamente divide el audio en fragmentos de unos 10 min y los va transcribiendo llamando al método *llamada_a_whisper(audio, lan, translate)*.

- *llamada_a_whisper(audio, lan, translate)*, a la vez que transcribe un solapamiento en los momentos en los que se fragmenta el audio. Una vez obtenido el texto del solapamiento se comparan ambos textos y se comprueba si la transcripción ha añadido texto de más al final de cada fragmento, para así eliminarlo, para después ir concatenado este texto en una variable que será lo que retorne el método.

- *transcribe(audio, lan, translate)*:Toma como parámetros el audio a transcribir, en stream de bytes, el lenguaje y un booleano que refleja si el usuario quiere traducir o transcribir el audio. Es el método principal de la clase. Básicamente se encarga de comprobar si el audio pesa más de 25MB. Si no es así llama al método *llamada_a_whisper(audio, lan, translate)*, y si pesa más de 25MB, entonces llama al método *transcribe_long(audio, lan, translate)*. Retorna el texto que devuelve o *llamada_a_whisper(audio, lan, translate)* o *transcribe_long(audio, lan, translate)*.

- *check_key(clave)*: se trata de un método que recibiendo como parámetro una clave devuelve un booleno indicando si esta es o no es válida para OpenAI.

Por lo tanto, ambas opciones devuelven el mismo resultado, que es el texto resultante de la transcripción del audio, ya sea obtenido por la API de Whisper o ejecutando Whisper en un servidor propio. Además, ambos son llamados por el fichero principal app.py, con el método *server.whisper(nombre_audio, lan, model, gpu, trad)* o *whisper.transcribe(audio, lan, translate)*.

3.3.3. Segmentación en párrafos

Este módulo finaliza la ejecución del proyecto con la segmentación del texto en párrafos. La llamada a la segmentación de párrafos se produce mediante el fichero chatgpt.py, que a su vez es llamado por el fichero principal app.py, a través del método *chatgpt.respuesta()*. Este módulo se encarga de hacer la petición a la API de chatgpt adaptando el texto a las necesidades de dicha API, es decir, dividiendo el texto para que no supere los 4096 tokens. Estos son los principales métodos:

- *pregunta(peticion)*: este método toma como parámetro una petición para GPT-3.5 y simplemente hace una llamada a su API y retorna la respuesta que da GPT-3.5

- *respuesta(texto)*: este método es el principal. Toma como parámetro el texto a segmentar en párrafos, el cual es dividido en fragmentos que no superen los 4096 caracteres sin dejar a la mitad una oración, usando NLTK [23]. Después se formula el promt y se llama al método *pregunta(peticion)* para cada fragmento. Por último, se van concatenando las respuestas en un String que es lo que retorna este método. Si el usuario no ha seleccionado la opción de segmentación no ejecuta este método.

3.3.4. Visualización de resultados

Este módulo muestra los resultados de la ejecución del proyecto. Para mostrar por pantalla no se requiere de ningún módulo. Sin embargo, para enviar los resultados por correo si se necesita.

La llamada al envío de los resultados por correo se produce mediante el fichero correo.py, que a su vez es llamado por el fichero principal app.py, a través del método *correo.send()*. Esta clase tiene los siguientes métodos:

- *check()*: comprueba que el correo electrónico que se le pasa por parámetro es correcto.
- *send()*: este método recibe como parámetro el nombre del audio que ha seleccionado el usuario, el texto a enviar y la dirección de correo del receptor. Con esto envía un correo al receptor.

3.4. Herramientas utilizadas

Este proyecto ha sido desarrollado por completo en Python 3.10 en una máquina de sistema operativo Windows 10, con Pycharm Community Edition como entorno de desarrollo. Además, se ha usado un servidor externo con varias GPUs para ejecutar algunos procesos de la aplicación y un servidor más pequeño para alojar la aplicación.

Entre las distintas librearías empleadas que utiliza la aplicación web, cabe destacar:

- Pytorch: librería de código que resulta indispensable para Pyannote.Audio, pues es una librería para el entrenamiento y uso de redes neuronales [24].

- Ffmpeg: es un módulo usado en el proyecto para el análisis del audio, más concretamente se usa para decodificar el audio y, independientemente de la extensión del archivo de audio, transformarlo a cualquier formato, como .WAV [25].
- Streamlit: Utilizada para el desarrollo rápido e intuitivo de aplicaciones web en Python, en código abierto, permitiendo transformar scripts en aplicaciones web en minutos [16].
- NLTK o Natural Language Toolkit: es una plataforma que permite trabaja con datos del lenguaje humano, es decir, que se trata de una librería de procesamiento de lenguaje natural (NLP), por lo que proporciona una gran variedad de herramientas y recursos para el análisis y manipulación de textos [23].
- Paramiko: se trata de una herramienta de Python que implementa el protocolo SSH, lo cual le permite proveer las funcionalidades de cliente y servidor, lo que permite acceder a la consola para subir archivos y acceder al servidor propio para ver el contenido de los ficheros generados por la ejecución de Whisper [26].
- Pydub.AudioSegment: permite manipular y dividir archivos de audio así como exportar esos fragmentos en otros formatos, todo esto con una interfaz simple [27].
- openai: se trata de la librería de OpenAI en Python que provee acceso a la API de OpenAI para aplicaciones desarrolladas en Python. Con esta librería se consigue acceder de esta manera tanto a la API de Whisper como a la de GPT-3.5, pues ambas pertenecen a OpenAI [28].
- Pyannote.audio: es una herramienta de código abierto desarrollada en Python para la diarización de interlocutores. Esta está basada en el framework de Machine Learning de Pytorch, y provee un conjunto de bloques de construcción de redes neuronales entrenables que pueden ser combinados y optimizados de manera conjunta para construir pipelines de diarización de interlocutores. Además, posee modelos pre-entrenados que aceleran mucho el desarrollo de aplicaciones [5].

3.5. Pruebas y resultados

Para evaluar el resultado final del texto generado por la herramienta, se va a comparar el texto de un audio transcrito a mano y el generado automáticamente por esta herramienta. Antes de nada, mencionar que este proyecto no se centra en la precisión de la diarización ni de la transcripción, si no que se centra en integrar los 3 modelos de IA para producir un texto legible.

Este es el texto resultante de transcribir el audio a mano:

[Speaker 0]: Secretaría de Madrid Moda, buenos días. Me llamo Helena Fernández, ¿en qué puedo ayudarle?

[Speaker 1]: Buenos días, mi nombre es Carlos Molinero y me gustaría hablar con don Fernando Torres, por favor.

[Speaker 0]:

Si, por su puesto, le paso con su despacho

Perdone la espera, pero por desgracia en estos momentos el señor Torres no puede atenderle porque no se encuentra en la oficina. Volverá por la tarde.

[Speaker 1]: ¿A que hora vuelve por favor?

[Speaker 0]: Por la tarde estará en su oficina después de las cuatro.

[Speaker 1]: Vale, puedo llamar después de las cuatro.

[Speaker 0]: Si, por supuesto. Pero si quiere también puede dejar un mensaje.

[Speaker 1]:

Si, creo que será mejor.

Puede, por favor decirle al señor Torres que le ha llamado Carlos Molinero. Le llamo en relación a la oferta de trabajo en el departamento de Recursos Humanos.

[Speaker 1]:

Muy bien, dejo el mensaje al señor Torres.

¿Puede darme su número de teléfono por favor?

[Speaker 1]: Si, es el 655274872.

[Speaker 0]: Muchas gracias, se lo repito. Es el 655274873.

[Speaker 1]: No, es 655274872, el último número es 2, no 3.

[Speaker 0]: Bien, gracias, se lo repito otra vez para estar segura: 655274872.

[Speaker 1]: Si, ahora está correcto.

[Speaker 0]: Podría deletrearme su apellido por favor.

[Speaker 1]: Es M-O-L-I-N-E-R-O.

[Speaker 0]: Gracias, le diré al señor Torres que ha llamado para el trabajo y que puede ponerse en contacto con usted en el 655274872.

[Speaker 1]: Si, muy bien. Muchas gracias es uste muy amable.

[Speaker 0]: De nada, adiós

[Speaker 1]: Adiós.

Figura 3.12. Ejemplo de transcripción de audio con diarización de interlocutores y segmentación de párrafos generado a mano.

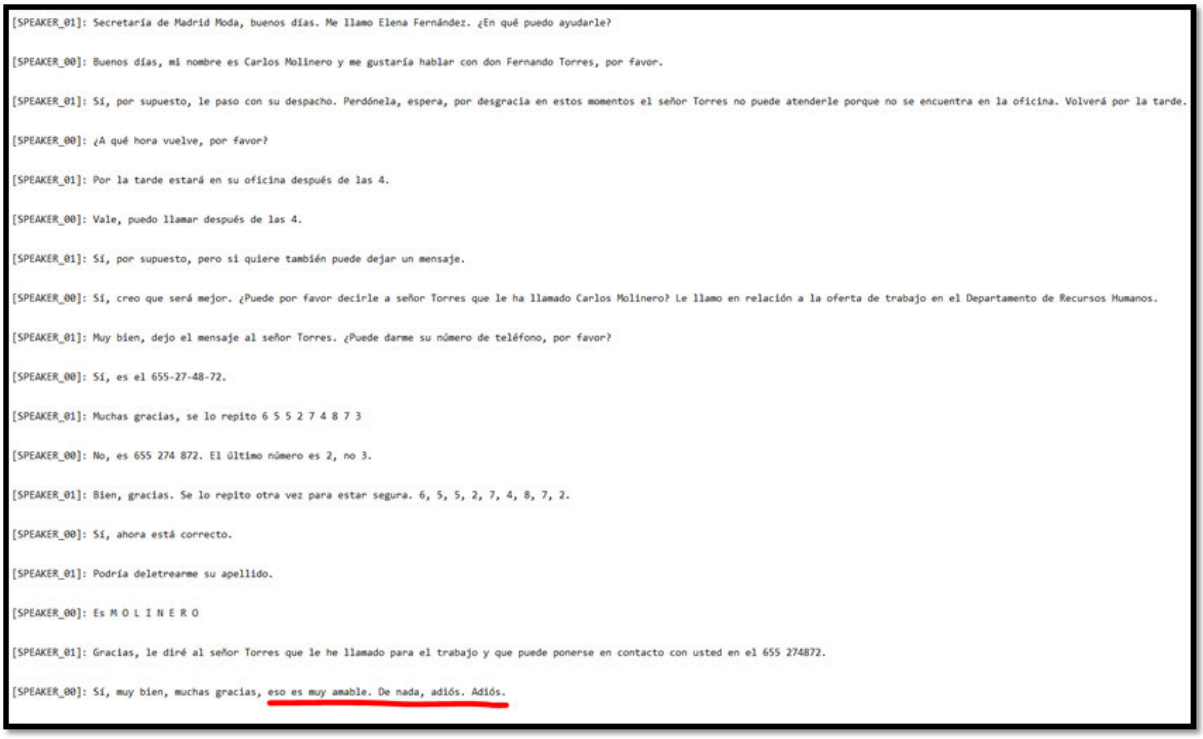
Como se puede ver en la Figura 3.12, se trata de una transcripción de un audio con diarización de interlocutores y segmentación en párrafos la cual a sido generada por un ser humano sin usar Inteligencia Artificial.

Trasncripción:

Buenos días, mi nombre es Carlos Molinero y me gustaría hablar con don Fernando Torres, por favor. Sí, por supuesto, le paso con su despacho. Perdónela, espera, por desgracia en estos momentos el señor Torres no puede atenderle porque no se encuentra en la oficina. Volverá por la tarde. ¿A qué hora vuelve, por favor? Por la tarde estará en su oficina después de las 4. Vale, puedo llamar después de las 4. Sí, por supuesto, pero si quiere también puede dejar un mensaje. Sí, creo que será mejor. ¿Puede por favor decirle al señor Torres que le ha llamado Carlos Molinero? Le llamo en relación a la oferta de trabajo en el Departamento de Recursos Humanos. Muy bien, dejo el mensaje al señor Torres. ¿Puede darme su número de teléfono, por favor? Sí, es el 655 27 48 72. Muchas gracias, se lo repito, 655 27 48 73. No, es 655 27 48 72. El último número es 2, no 3. Bien, gracias, se lo repito otra vez para estar segura, 655 27 48 72. Sí, ahora está correcto. ¿Podría deletrearme su apellido, por favor? Es M O L I N E R O. Gracias, le diré al señor Torres que le ha llamado para el trabajo y que puede ponerse en contacto con usted en el 655 27 48 72. Sí, muy bien, muchas gracias, eso es muy amable. De nada, adiós. Adiós.

Figura 3.13. Ejemplo de transcripción de audio sin diarización de interlocutores ni segmentación de párrafos generado por la herramienta.

La figura 3.13 refleja una transcripción del mismo audio que la figura 3.12, pero usando la herramienta, sin segmentación en párrafos ni diarización de hablantes. Como se puede ver en ambas figuras el texto que contienen es el mismo pero representado de manera diferente. Esto nos indica que la herramienta se integra de manera adecuada con el módulo de Whisper. Sin embargo, pueden existir algunos audios que añadan texto de más al final de la transcripción en algunas transcripciones. Esto sucede en audios con mucho ruido de fondo, como puede ser el ventilador del ordenador o el tráfico, los cuales tienen momentos en los que no hay ningún interlocutor hablando, pero el ruido permanece de fondo, por lo que el modelo de Whisper lo detecta como que alguien está hablando, pero no termina de reconocer lo que dicen, y en base al contexto del texto generado se rellena palabras, o directamente mete palabras repetidas y sin sentido.



[SPEAKER_01]: Secretaría de Madrid Moda, buenos días. Me llamo Elena Fernández. ¿En qué puedo ayudarle?

[SPEAKER_00]: Buenos días, mi nombre es Carlos Molinero y me gustaría hablar con don Fernando Torres, por favor.

[SPEAKER_01]: Sí, por supuesto, le paso con su despacho. Perdónela, espera, por desgracia en estos momentos el señor Torres no puede atenderle porque no se encuentra en la oficina. Volverá por la tarde.

[SPEAKER_00]: ¿A qué hora vuelve, por favor?

[SPEAKER_01]: Por la tarde estará en su oficina después de las 4.

[SPEAKER_00]: Vale, puedo llamar después de las 4.

[SPEAKER_01]: Sí, por supuesto, pero si quiere también puede dejar un mensaje.

[SPEAKER_00]: Sí, creo que será mejor. ¿Puede por favor decirle a señor Torres que le ha llamado Carlos Molinero? Le llamo en relación a la oferta de trabajo en el Departamento de Recursos Humanos.

[SPEAKER_01]: Muy bien, dejo el mensaje al señor Torres. ¿Puede darme su número de teléfono, por favor?

[SPEAKER_00]: Sí, es el 655-27-48-72.

[SPEAKER_01]: Muchas gracias, se lo repito 6 5 5 2 7 4 8 7 3

[SPEAKER_00]: No, es 655 274 872. El último número es 2, no 3.

[SPEAKER_01]: Bien, gracias. Se lo repito otra vez para estar segura. 6, 5, 5, 2, 7, 4, 8, 7, 2.

[SPEAKER_00]: Sí, ahora está correcto.

[SPEAKER_01]: Podría deletrearme su apellido.

[SPEAKER_00]: Es M O L I N E R O

[SPEAKER_01]: Gracias, le diré al señor Torres que le he llamado para el trabajo y que puede ponerse en contacto con usted en el 655 274872.

[SPEAKER_00]: Sí, muy bien, muchas gracias, eso es muy amable. De nada, adiós. Adiós.

Figura 3.14. Ejemplo de transcripción de audio con diarización de interlocutores, pero sin segmentación de párrafos generado por la herramienta.

La figura 3.14 muestra una transcripción del mismo audio que las otras dos figuras anteriores, pero esta vez incluye la diarización de interlocutores y ha sido generada por la herramienta. Al compararlo con las figuras 3.13 y 3.12 podemos ver que el contenido de la transcripción es el mismo. Otra conclusión que se puede sacar es que la diarización del modelo pre-entrenado de pyannote.Audio es bastante precisa, aunque, cabe destacar que no es infalible. Como se puede ver en la figura 3.14, la parte subrayada en rojo es donde la diarización falla, probablemente se deba a que es un inter-

cambio corto de palabras que le dificulta reconocer quien habla y por ello no distinga un cambio de interlocutor. Una vez comparado la figura 3.14 con las anteriores transcripciones podemos concluir que el módulo de la diarización funciona correctamente y se integra perfectamente con el módulo de transcripción.

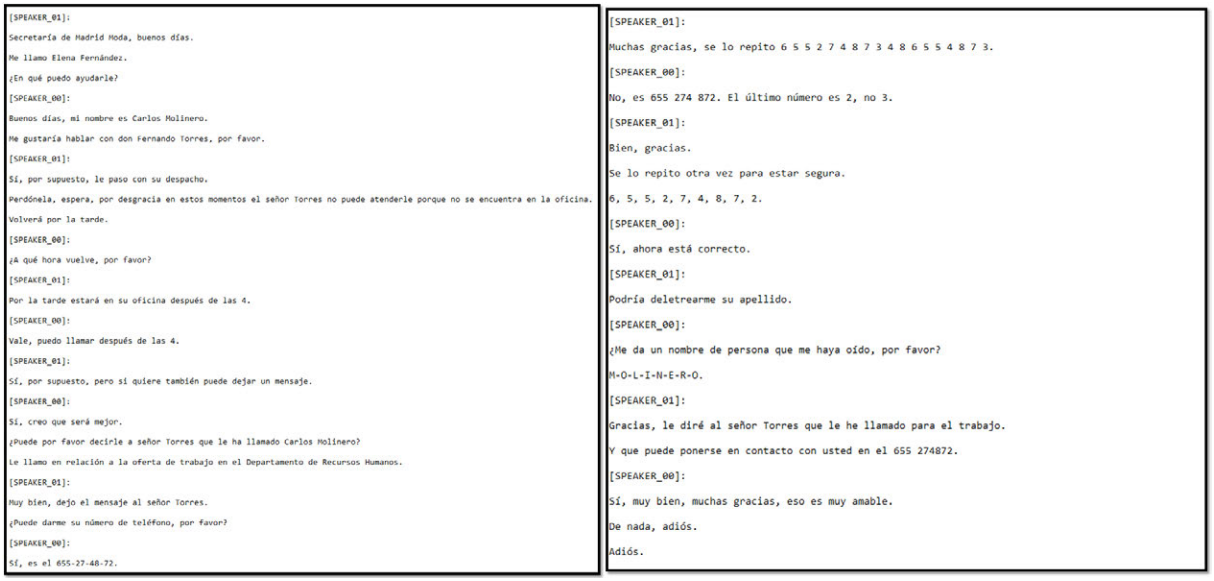


Figura 3.15. Ejemplo de transcripción de audio con diarización de interlocutores, pero sin segmentación de párrafos generado por la herramienta.

Por último, la figura 3.15 muestra una transcripción del audio que incluye diarización de interlocutores y segmentación en párrafos, llevado a cabo por la herramienta. Comparándolo con las tres figuras anteriores resalta que el contenido de la transcripción es el mismo, y también, las intervenciones por la diarización de interlocutores son las mismas que en la figura 3.14. Sin embargo, la segmentación en párrafos realizada por la herramienta difiere de la realizada por un ser humano. Esto se debe a que GPT-3.5 es estocástico e indeterminista, es decir, cada vez que se le pregunte al modelo una misma pregunta, su respuesta variará en algo, aunque sea un detalle muy pequeño, por mucho que la pregunta le limite la capacidad de autogenerar marcándole las pautas.

En base a todas estas comparaciones se puede concluir que los tres modelos se integran de manera correcta entre sí. Cabe recordar que el propósito de este proyecto no se centra en la precisión de la diarización ni de la transcripción, más bien se centra en la integración de los 3 modelos de IA para producir un texto legible.

3.6. Impacto social y medioambiental

Los textos generados, anteriormente descritos, mejoran de forma drástica la comprensión y visualización de transcripciones de audio, pues, aunque si es cierto que ya existen apps de transcripción de audios, ninguna de ellas ofrece unos resultados tan legibles, con segmentación en párrafos y diarización de interlocutores, como lo hace esta herramienta.

Esto ayudaría a cualquier usuario que desee transcribir un audio, ya que, aparte de tenerlo en texto, siempre interesa que dicho texto resulte fácil de leer, sobre todo para los textos largos.

En muchas ocasiones, en el trabajo, se graban reuniones para compañeros que no pueden acudir a estas reuniones, pero después les puede costar rengancharse, por lo que una transcripción legible y sencilla ayuda a comprender más rápidamente dicha reunión. Esto permitiría a estos trabajadores atender a las próximas reuniones estando al día de todos los asuntos tratados en las reuniones transcritas.

También ayudaría a estudiantes, que quieran realizar apuntes de la clase anterior, pues con estas transcripciones comprenden más rápidamente la información, lo cual mejoraría el rendimiento académico de los alumnos.

Además, este proyecto permitiría acercar la Inteligencia Artificial a los usuarios que no están dentro del mundo de la IA, permitiendo que se familiaricen con los conceptos básicos de esta.

En cuanto al impacto medioambiental, la utilización de este proyecto reduciría significativamente el uso de papel al tomar apuntes ya sea en una reunión o una clase. Normalmente, para tomar apuntes se usan varias hojas de papel, sobre todo en el caso de los estudiantes. Este gasto de papel se reducirá considerablemente ya que, con el audio de la reunión, clase u otro tipo de grabaciones, puede generar una transcripción fácil de leer y comprender por lo que los apuntes no serán tan necesarios, y por tanto no hará falta tanto papel.

Sin embargo, también existen desventajas en este ámbito. Al ejecutar en GPU se consume una gran cantidad de energía. Esto resulta perjudicial para el medio ambiente.

3.7. Metodología utilizada

Para el desarrollo de este proyecto se ha utilizado una metodología con prototipado. Utilizando esta metodología, durante la duración del proyecto se han desarrollado un total de 7 prototipos distintos hasta obtener el sistema final:

- El primer prototipo se trataba simplemente de un "Hola mundo", es decir, levantar la aplicación web, pero sin que esta contuviese nada más que un pequeño texto inicial. El feedback del usuario para este prototipo fue positivo, y pidió que a continuación se implementase la transcripción del audio usando Whisper.
- El segundo prototipo ya fue bastante más elaborado que el primero, ya que se implementó una opción para seleccionar el audio a transcribir usando Whisper, un menú para elegir la configuración para ejecutar Whisper y la funcionalidad de transcribir dicho audio con la configuración seleccionada. Sin embargo, solo se implementó la ejecución de Whisper en un servidor propio, por ello, una vez presentado este feedback al usuario, este sugirió que se añadiese una opción para ejecutarlo a través de la API de Whisper.
- El tercer prototipo consistió en implementar la transcripción del audio mediante la API de Whisper. Una vez implementado, se presentó el prototipo al usuario y este pidió que se implementase la segmentación en párrafos.
- El cuarto prototipo consiste en desarrollar el módulo de segmentación en párrafos e integrarlo con el módulo de transcripción. Al ser presentado al usuario, este pidió que, al mostrar el resultado, el usuario tuviese una opción de visualizar el texto sin segmentar en párrafos.
- En el quinto prototipo se implementó la opción de visualizar el texto con y sin segmentación de párrafos si esta opción había sido marcada por el usuario. El feedback sobre este prototipo resultó ser positivo. Sin embargo, el usuario pidió que se implementase a continuación la diarización de interlocutores.
- Durante el sexto prototipo se desarrolló el módulo de diarización de interlocutores y se integró con el resto de módulos. En este caso el feedback también fue positivo y el usuario pidió que se incluyese que, al terminar todo el proceso, el sistema enviase un correo con los resultados.
- El séptimo prototipo fue el último, en el cual se llevó a cabo la implementación del envío de los resultados al usuario por correo electrónico al acabar el proceso. El feedback en esta fase fue positivo y este prototipo resultó aprobado por el usuario.

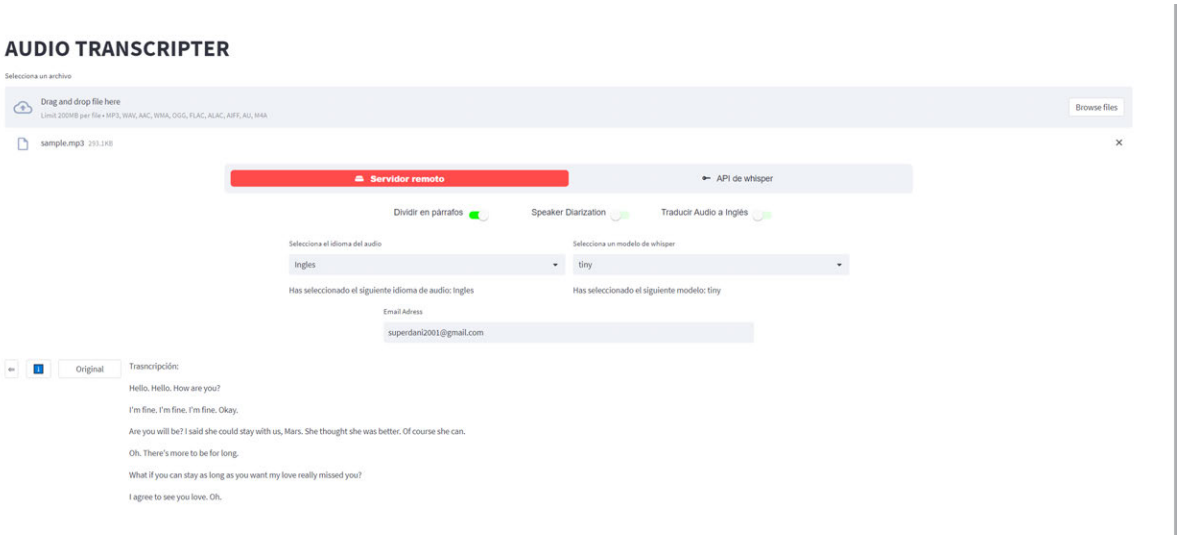


Figura 3.16. Ejemplo del resultado de la aplicación web.

En la figura 3.16 se puede ver el séptimo y último prototipo, que al ser el prototipo final, aprobado por el cliente, se corresponde con la versión final de la aplicación web.

4. Conclusiones y trabajos futuros

4.1. Conclusiones

Durante el desarrollo de este proyecto, han surgido algunas complicaciones. Una de ellas ha sido la implementación de la transcripción a través de la API de Whisper ya que, como esta no permite audios superiores a 25MB, hubo que recortar los audios pesados en fragmentos de 10 minutos cada uno, empleando la biblioteca pydub.AudioSegment. Esta biblioteca provocaba conflictos con la versión instalada de Ffmpeg, lo cual retrasó significativamente el desarrollo del proyecto. Otra de las complicaciones que han surgido a lo largo del desarrollo del proyecto fue la integración del módulo de diarización de interlocutores con el módulo de transcripción de audios. Esto se debe a que cuando se desarrolló el módulo de transcripción no se puso ningún módulo de conversión del audio a .WAV, pues Whisper acepta todo tipo de extensiones de audio. Sin embargo, la biblioteca de pydub.Audio es más específica en cuanto a los tipos de audio que admite, pues solo funciona con un .WAV. Por ello hubo que añadir al principio del desarrollo del proyecto un módulo de conversión de audio a .WAV. Por otro lado, los siguientes objetivos han sido cumplidos satisfactoriamente:

- La correcta diarización de las intervenciones de los interlocutores, en el menor tiempo posible. Esta es una de las partes clave para poder generar un texto claro y legible.
- Transcribir un audio a texto todo lo rápido que se pueda, indispensable si se quiere obtener un texto de un audio.
- Segmentar el texto resultante de la transcripción. Esta es otra de las claves para lograr un texto claro y legible.
- Desarrollo de una aplicación web en la que el usuario pueda decidir si ejecutar whisper desde un servidor propio o desde la API de Whisper, importante a la hora de acelerar la transcripción de un audio.

- Desarrollar una aplicación web que pueda realizar todo este proceso de diarización, transcripción y segmentación en párrafos con todo tipo de archivos de audio como .mp3, .m4a, .WAV, etc. Este requisito es fundamental para acercar la IA a todos los usuarios, no solo los expertos en el mundo de la informática.

Sin embargo, el objetivo de la segmentación en párrafos, aunque se ha logrado, los resultados pueden mejorarse, ya que no se ha logrado que los resultados, para un mismo texto, sean siempre igual. Sin embargo, si que se han sentado las bases para su desarrollo en un futuro.

Gracias al cumplimiento de los objetivos, esta herramienta puede ayudar a muchos usuarios de cualquier ámbito, ya sea laboral, profesional o personal a obtener de un audio textos legibles y comprensibles.

Cabe recalcar que se ha habilitado un repositorio en GitHub con el código íntegro del desarrollo de la herramienta de forma que cualquier persona pueda descargarlo y mejorarlo [22].

4.2. Líneas Futuras

Este proyecto abre las puertas a varias líneas de seguimiento y de mejora posibles.

La principal sería la mejora del módulo de segmentación en párrafos. Actualmente la cantidad de herramientas de segmentación en párrafos es limitada, por lo que resulta más interesante usar un modelo lenguaje para este tipo de tareas como es GPT-3.5. Lo ideal sería encontrar una herramienta que mejorase las prestaciones actuales del módulo de segmentación en párrafos. Otra posible vía de desarrollo para mejorar las prestaciones de dicho módulo sería la de ajustar el promt para que GPT-3.5 genere una respuesta consistente.

En cuanto al modelo utilizado, se podría mejorar el modelo utilizado, pues ya se encuentra disponible GPT-4, un modelo mucho más potente que GPT-3.5. Sin embargo, durante el desarrollo del proyecto, este modelo no ha estado disponible todavía para todos los usuarios, si no que había que apuntarse a una lista de espera.

Otra posible línea de mejora es la de incluir una opción en la aplicación de transcribir audios de YouTube, simplemente utilizando el link del video. Para este caso los pasos a seguir para implementar esta solución están más claros, puesto que en la actualidad existen herramientas como pytube que permiten descargar videos de esta plataforma desde código Python. Esta línea de mejora resulta mu-

cho más sencilla que la anterior y resulta una gran manera de acercar mucho más esta herramienta al ámbito profesional, académico y personal de los usuarios.

Por último, podría estudiarse una manera de acelerar la ejecución de Whisper en el servidor propio, ya que, aunque no es muy lento, en comparación con la velocidad de ejecución de la API de Whisper, sí que parece realmente lento. Esto ayudaría a reducir el coste al usuario de usar la herramienta ya que OpenAI cobra cada vez que se le hace una llamada a la API de Whisper.

Bibliografía

[1] I. vasco de administración pública, «La importancia de los párrafos,» 2017. dirección: https://www.ivap.euskadi.eus/entrada-blog/2017/la-importancia-de-los-parrafos-1/z16-b2aha/es/.

[2] S. PAUN DE GARCÍA, *Manual de investigación literaria: como preparar informes, trabajos de investigación, tesis y tesinas*. Castalia, 2004, págs. 125-131.

[3] D. CASSANY, *La cocina de la escritura*. Anagrama, 1999, págs. 82-93.

[4] H. BREDIN, «pyannote audio: neural building blocks for speaker diarization,» 2020. dirección: https://www.youtube.com/watch?v=37R_R82lfwA&ab_channel=Herv%C3%A9Bredin.

[5] H. BREDIN, «PYANNOTE.AUDIO 2.1 SPEAKER DIARIZATION PIPELINE: PRINCIPLE, BENCHMARK, AND RECIPE,» dirección: http://catedrartve.unizar.es/reto2022/PYA_report.pdf.

[6] GitHub, «Pyannote.Audio respository: Models,» dirección: https://github.com/pyannote/DEPRECATED-pyannote-audio-hub#models.

[7] OpenAI, «Introducing Whisper,» dirección: https://openai.com/research/whisper.

[8] C. S. Franco, «Mel Cepstral Frequency Coefficients MFCC,» dirección: https://francocarlos.com/2017/05/04/mel-cepstral-frequency-coefficients-mfcc/.

[9] OpenAI, «Available models and languages,» dirección: https://github.com/openai/whisper#available-models-and-languages.

[10] OpenAI, «Introducing ChatGPT and Whisper APIs,» dirección: https://openai.com/blog/introducing-chatgpt-and-whisper-apis.

[11] Sciforce, «What is GPT-3, How Does It Work, and What Does It Actually Do?,» dirección: https://medium.com/sciforce/what-is-gpt-3-how-does-it-work-and-what-does-it-actually-do-9f721d69e5c1.

[12] A. Romero, «Understanding GPT-3 In 5 Minutes,» dirección: https://towardsdatascience.com/understanding-gpt-3-in-5-minutes-7fe35c3a1e52.

[13] H. Koshti, «Understanding the GPT-3.5 Architecture,» dirección: https://www.linkedin.com/pulse/chatgpts-guide-understanding-gpt-35-architecture-heena-koshti/.

[14] opengenius, «GPT-3.5 model architecture,» dirección: https://iq.opengenus.org/gpt-3-5-model/.

[15] dev.to, «Explorando el GPT3.5-Turbo-16K: Una nueva frontera en la IA conversacional,» dirección: https://dev.to/makiai/explorando-el-gpt35-turbo-16k-una-nueva-frontera-en-la-ia-conversacional-3po2.

[16] S. Wiki, «Welcome to streamlit,» dirección: https://github.com/streamlit/streamlit#welcome-to-streamlit-.

[17] latentview, «Introduction to Streamlit,» dirección: https://www.latentview.com/data-engineering-lp/introduction-to-streamlit/.

[18] D. S. y Carmen Gerea, «Prototipo: qué es y para qué sirve,» dirección: https://freed.tools/blogs/ux-cx/prototipo.

[19] I. Carrillo, «Modelos de desarrollo Software,» dirección: https://freed.tools/blogs/ux-cx/prototipo.

[20] M. Martin, «Prototype Model in Software Engineering,» dirección: https://www.guru99.com/software-engineering-prototyping-model.html.

[21] Docker, «Docker documentation,» dirección: https://docs.docker.com/.

[22] GitHub, «Repositorio de la herramienta,» dirección: https://github.com/danielsalgadoinfantes/st_docker_template.

[23] NLTK, «Documentation,» dirección: https://www.nltk.org/#natural-language-toolkit.

[24] Pytorch, «Documentation,» dirección: https://github.com/pyannote/pyannote-audio#neural-speaker-diarization-with-pyannoteaudio.

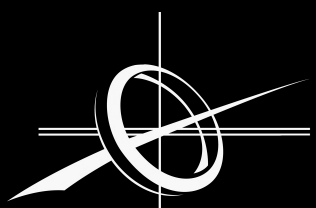
[25] ffmpeg, «About,» dirección: https://ffmpeg.org/about.html.

[26] Paramiko, «Welcome to paramiko,» dirección: https://www.paramiko.org/#welcome-to-paramiko.

[27] pydub.AudioSegment, «pydub 0.25.1,» dirección: https://pypi.org/project/pydub/.

[28] OpenAI, «OpenAI Python library,» dirección: https://github.com/openai/openai-python#openai-python-library.

Índice de términos



Universidad Politécnica de Madrid

ETSI **SISTEMAS INFORMÁTICOS**