Universidad Zaragoza
1542

Trabajo Fin de Grado

Desarrollo de Sistemas de Síntesis de Voz con Énfasis en la Prosodia
Development of Voice Synthesis Systems with Emphasis on Prosody

Autor/es

Jaime Domper Cerrada

Director/es

José Manuel Sánchez Aquilué

Grado en Ingeniería Informática

ESCUELA DE INGENIERÍA Y ARQUITECTURA

2024

Desarrollo de Sistemas de Síntesis de Voz con Énfasis en la Prosodia

Resumen

Este trabajo surgió a partir de un periodo de prácticas en el departamento de I+D de **ETIQMEDIA** [1], enfocado en el desarrollo de un sistema de doblaje automático multilingüe que respete las características prosódicas humanas. Los recientes avances en la síntesis de voz utilizando redes neuronales han permitido la creación de voces casi indistinguibles de las humanas, y este proyecto busca aprovechar esos avances.

Los objetivos principales fueron asegurar la sincronización precisa del audio doblado con el vídeo original, generar una voz que sea percibida como humana, producir doblaje en varios idiomas y diseñar un sistema modular para facilitar su mantenimiento y actualización.

Se implementó un diseño modular compuesto por varios módulos específicos. La transcripción se realizó utilizando la librería *Whisper* [6] de *OpenAI* [7] para obtener transcripciones precisas con marcas de tiempo. La traducción se llevó a cabo mediante la implementación de modelos *OPUS MT*[11], que permiten traducir las transcripciones a otros idiomas. Para la síntesis de voz (TTS), se utilizó el modelo *Xtts v2*[3], capaz de generar audio a partir de texto.

Los resultados demostraron que el sistema logró una alta precisión en la transcripción con modelos *Whisper* [6] con un 9.6 y 7.6 de WER en Español y en Inlgés respectivamente, buenas puntuaciones BLEU (58 de media) en la traducción con modelos *OPUS MT*[11], y una naturalidad en la síntesis de voz comparable a la voz humana mediante técnicas de conversión y clonado de voz. La sincronización del audio doblado con el vídeo original fue precisa, cumpliendo con los objetivos planteados. Las pruebas incluyeron métricas de precisión en transcripción y traducción, así como la percepción subjetiva de naturalidad en la voz generada, validando así la robustez y calidad del sistema desarrollado.

En conclusión, este trabajo ha cumplido con éxito sus objetivos, proporcionando una solución avanzada y eficaz para el doblaje automático con características prosódicas humanas y capacidades multilingües. El diseño modular asegura la sostenibilidad y escalabilidad del sistema, permitiendo futuras actualizaciones y mejoras. Este proyecto ha mejorado las capacidades de doblaje de **ETIQMEDIA** [1], incorporando nuevas tecnologías y metodologías que permiten una mayor calidad y naturalidad en la síntesis de voz.

Las posibles continuaciones incluyen el uso de modelos de lenguaje más avanzados para mejorar la traducción y la coherencia del doblaje, la implementación de técnicas de sincronización labial, y el desarrollo de interfaces gráficas para facilitar la corrección y edición manual del doblaje. Estas mejoras potenciales aseguran que el sistema pueda adaptarse y evolucionar con los avances tecnológicos.

[1] `https://etiqmedia.com/`

Agradecimientos

Considero oportuno añadir una breve reflexión acerca de lo que ha supuesto para mí, tanto en lo personal como en lo profesional, la realización de este trabajo.

Uno de los aspectos más enriquecedores de este proyecto ha sido el constante desafío que ha representado. Durante muchas semanas, me he acostado con un problema en mente, dándole vueltas hasta que finalmente encontré la solución. Esa sensación de satisfacción al encajar una pieza del puzzle que parecía perdida es lo que me ha motivado a seguir adelante con entusiasmo. Este proceso me ha enseñado la importancia de la perseverancia y la paciencia en la resolución de problemas complejos.

El hecho de que este proyecto haya sido de I+D me ha obligado a adentrarme en áreas en las que no me sentía cómodo y a salir de mi zona de confort. He tenido que investigar y aprender sobre tecnologías y metodologías que antes me eran desconocidas. Este desafío ha sido crucial para mi crecimiento profesional, ya que me ha permitido adquirir nuevas habilidades y conocimientos, y me ha mostrado la importancia de estar siempre dispuesto a aprender y adaptarse a nuevas situaciones.

Además, el trabajo en equipo ha sido una parte fundamental de este proyecto. He tenido la suerte de contar con compañeros de empresa que siempre estuvieron dispuestos a asistir sin demora, incluso fuera del horario laboral. Su apoyo y colaboración han sido invaluables, y estoy muy agradecido de haber formado parte de este equipo. Este ambiente de trabajo colaborativo me ha enseñado la importancia de la comunicación y la cooperación para alcanzar objetivos comunes.

Comencé mi carrera con el deseo de crear mis propias soluciones a los problemas que me encontrase. En aquel entonces, no tenía ni idea de programación ni de cómo enfrentarme a un problema complejo. Sin embargo, gracias a estos años de carrera y a la experiencia adquirida en este proyecto, puedo decir con certeza que eso ha cambiado. He adquirido las herramientas y la confianza necesarias para enfrentar y resolver problemas de manera efectiva.

En resumen, la realización de este trabajo ha sido una experiencia profundamente enriquecedora que ha influido significativamente en mi desarrollo personal y profesional. He aprendido a enfrentar desafíos con determinación, a colaborar efectivamente con otros, y a aplicar mis conocimientos para crear soluciones prácticas y útiles. Este proyecto no solo ha ampliado mis habilidades técnicas, sino que también ha fortalecido mi pasión por la tecnología y la innovación.

Índice

Glosario

API *Application Programming Interface*, es decir, Interfaz de Programación de Aplicaciones. Es un conjunto de reglas y protocolos que permite a diferentes programas de software comunicarse entre sí, facilitando la integración y el intercambio de datos.. 11, 15

ASR *Automatic Speech Recognition*, es decir, Reconocimiento Automático del Habla. Es una tecnología que permite a las computadoras identificar y procesar el habla humana en formato de audio, convirtiéndola en texto escrito.. 14, 16, 30

BLEU *Bilingual Evaluation Understudy*, es decir, Evaluación Bilingüe de Subordinado. Es una métrica utilizada para evaluar la calidad de texto generado por máquinas, como en la traducción automática, comparando el texto generado con uno o más textos de referencia.. 2, 4, 7, 17, 28, 36

chunks Bloques de datos o fragmentos. En el contexto del procesamiento de lenguaje natural, se refiere a segmentos de texto agrupados de acuerdo a ciertas reglas o características, como frases o cláusulas.. 11

GAN *Generative Adversarial Network*, es decir, Red Generativa Adversarial. Es un tipo de modelo de inteligencia artificial compuesto por dos redes neuronales enfrentadas entre sí: el generador y el discriminador.. 8

LLM *Large Language Model*, es decir, Modelo de Lenguaje Grande. Es un tipo de modelo de inteligencia artificial entrenado con grandes cantidades de datos textuales para comprender y generar lenguaje natural con un alto grado de coherencia y precisión.. 31

MT *Machine Translation*, es decir, Traducción Automática.. 10

NLP *Natural Language Processing*, es decir, Procesamiento de Lenguaje Natural. Es un campo de la inteligencia artificial que se enfoca en la interacción entre las computadoras y el lenguaje humano, permitiendo a las máquinas entender, interpretar y responder al lenguaje de manera efectiva.. 8, 10

threshold Umbral. En el contexto de la inteligencia artificial y el procesamiento de datos, se refiere a un valor límite que define un criterio para la toma de decisiones o la clasificación de datos.. 15

TTS *Text-to-Speech*, es decir, Conversión de Texto a Voz. Es una tecnología que convierte texto escrito en habla sintética, permitiendo que las computadoras y otros dispositivos electrónicos "hablen" en voz alta.. 2, 4, 8, 9, 11, 14, 20, 22, 24, 25, 29, 30, 32, 33

WER *Word Error Rate*, es decir, Tasa de Error de Palabras. Es una métrica utilizada para evaluar la precisión de los sistemas de reconocimiento automático del habla, calculando el porcentaje de palabras incorrectamente reconocidas en relación con el total de palabras pronunciadas.. 2, 4, 27, 29, 30, 38

Índice de figuras

Índice de cuadros

1. Introducción

A lo largo de los últimos años con la llegada llegada de las redes GAN se han conseguido grandes avances en la síntesis de voz. Actualmente los resultados bastante realistas en multitud de idiomas pero en ocasiones la salida de los modelos estándar no terminan de conseguir una prosodia de calidad similar a la de un ser humano. En este trabajo se ha construido un sistema de síntesis de audio multilingüe a partir de texto que cuide todos los elementos prosódicos.

El programa elaborado ha sido puesto a prueba en el caso de uso de doblaje automático, donde la voz tiene que ser lo más fiel a la original y coordinarse con los elementos visuales y acústicos de la escena.

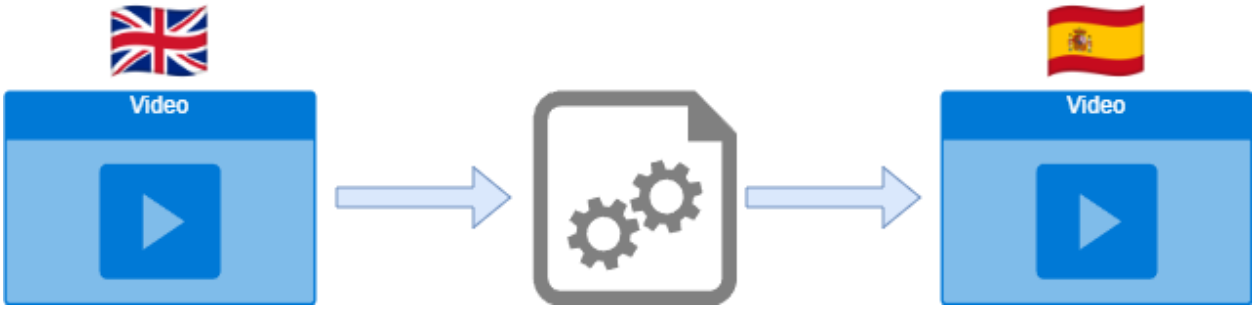

Figura 1: Idea general del programa deseado con ejemplo (de inglés a español)

1.1. Contexto y Motivación

Este trabajo ha sido realizado en empresa, mas concretamente en ETIQMEDIA, en el departamento de I+D. Dentro del departamento de I+D hay tres subgrupos:

- Audio
- NLP
- Vídeo

Ha sido un trabajo llevado a cabo dentro del equipo de NLP, aunque gran parte está relacionada con Audio y algo de Vídeo.

El doblaje de contenido audiovisual y la síntesis de texto a voz (TTS, por sus siglas en inglés) han experimentado una evolución significativa en los últimos años. Desde los primeros intentos de generar voz sintética con sonidos pregrabados hasta los avances actuales impulsados por inteligencia artificial y aprendizaje profundo, la calidad y naturalidad del TTS ha mejorado notablemente. En los años 70 y 80, los sistemas de TTS utilizaban concatenación de unidades fonéticas, lo que resultaba en una voz robótica y poco natural. A partir de los años 90, con la introducción de modelos estadísticos como HMM (Hidden Markov Models), se lograron mejoras considerables en la fluidez del habla sintetizada.

En la última década, el uso de redes neuronales y, más específicamente, de modelos como *WaveNet* de Google y Tacotron ha revolucionado el campo. Estos modelos han permitido la generación de voces que son casi indistinguibles de las humanas, gracias a su capacidad para aprender las complejidades del habla, incluyendo la prosodia, la entonación y el ritmo.

1.1.1. Doblaje y Prosodia

El doblaje es el proceso de sustituir las voces originales de una producción audiovisual por otras en un idioma diferente, manteniendo la sincronía con los movimientos labiales de los actores. Un aspecto crucial del doblaje es la **isocronía**, que se refiere al ritmo del habla en el original y en el doblado. Este término se deriva de las palabras griegas "iso"(igual) y "chronos"(tiempo), y se utiliza para asegurar que el tiempo de las pausas, la duración de las frases y el ritmo del habla en el doblaje coincidan con el original.

La isocronía es esencial para mantener la naturalidad y la credibilidad del doblaje. Por ejemplo, si en el vídeo original hay una pausa de 1,58 segundos, en el vídeo doblado también debe existir dicha pausa para que las imágenes y el audio estén perfectamente sincronizados. Esta sincronía no solo ayuda a mantener la coherencia temporal, sino que también contribuye a la percepción emocional del contenido, ya que el ritmo y las pausas juegan un papel fundamental en la comunicación de sentimientos e intenciones.

En conclusión, tanto la síntesis de voz como el doblaje dependen en gran medida de la capacidad para replicar de manera precisa las características prosódicas del habla original. Los avances en TTS y las técnicas de doblaje modernas buscan precisamente este objetivo, utilizando tecnología avanzada para producir resultados que sean lo más naturales y fieles posibles al material original.

1.2. Objetivos

El objetivo principal ha sido conseguir un sistema capaz de sintetizar voz con características prosódicas similares a las de un humano y para ello se ha elegido el caso de prueba de doblaje automático.

Se han enumerado los siguientes objetivos principales:

1. Conseguir una buena sincronización del audio doblado sobre el vídeo original, tarea que deberá ser llevada a cabo por el módulo de *Traducción* en su mayor parte.
2. Generar una voz que pueda ser interpretada como humana, poniéndola a prueba con una evaluación de tipo encuesta.
3. Doblar en más de un idioma, es decir, hacer que el sistema sea multilenguaje.
4. Diseñar sistema modular que sea sencillo de mantener y actualizar

1.3. Estado del Arte

Se ha llevado a cabo un análisis previo para conocer otras aproximaciones al mismo problema. Tras diseñar una arquitectura propia como posible solución basándose en arquitecturas de otros trabajos como [10], se ha ido módulo a módulo, tratando cada uno como un problema independiente a resolver.

La arquitectura básica en doblaje suele consistir en tres módulos:

- Módulo de transcripción
- Módulo de traducción
- Módulo de TTS

El módulo de transcripción es crucial para convertir el audio original del vídeo en texto. Una de las tecnologías más avanzadas en este campo es *Whisper de OpenAI*, que utiliza modelos de lenguaje basados en redes neuronales para proporcionar transcripciones precisas con marcas de tiempo.

Para la traducción de las transcripciones a otros idiomas, se emplean modelos como *OPUS MT*,

que están optimizados para mantener la coherencia y el contexto del texto original. Estos modelos utilizan técnicas avanzadas de procesamiento de lenguaje natural (NLP) para asegurar que la traducción sea precisa y fluida. Existen modelos de lenguaje mas grandes entrenados por la empresa *facebook* que también lideran el campo de MT.

En la última década, los avances en TTS han sido significativos, especialmente con la introducción de modelos como *WaveNet* y *Tacotron*. Estos modelos generan voz casi indistinguible de la humana, aprendiendo las complejidades del habla, incluyendo la prosodia, la entonación y el ritmo. El modelo Xtts v2 es un ejemplo destacado, capaz de generar audio a partir de texto con una naturalidad impresionante.

A la arquitectura básica se han añadido módulos adicionales para mejorar la calidad y la precisión del doblaje:

- Módulo de separación de audio: Separa las voces del fondo musical para aislar el diálogo. Permite reemplazar la voz por completo en vez de bajar el volumen de la original.
- Conversión de voz: Transfiere las características del hablante original al nuevo audio generado sin necesidad de entrenamiento extenso del modelo (aproximadamente 6 segundos de audio son suficientes).
- Clonado de voz: Utilizado para obtener un resultado lo más fiel posible al original, aunque requiere entrenamiento específico del modelo para la voz del hablante, no siendo viable para operaciones "*on the go*".

Estos módulos adicionales permiten una mayor flexibilidad y precisión en el doblaje, facilitando la adaptación de la voz sintetizada a las características específicas del hablante original, lo que resulta en una experiencia más natural y auténtica para el usuario.

1.4. Organización de la memoria

El problema se ha dividido en módulos integrados en un *pipeline*. La memoria de este trabajo también se estructura de una forma similar. Las siguientes dos secciones se dividen en cuatro subseciones para explicar cada uno de los módulos principales del programa.

Primero se detallan los métodos, el diseño y las decisiones de implementación llevadas acabo para cada uno de los módulos, especificando algoritmos más importantes y los pasos que se realizan en cada uno de los módulos.

Después se comenta como se ha ido evaluando cada modulo de forma independiente y el doblaje en su conjunto y se han comentado los resultados obtenidos.

En la ultima sección se han detallado las conclusiones del trabajo, haciendo una valoración del cumplimiento de los objetivos iniciales y reflexionando sobre las aportaciones del trabajo.

2. Métodos

Se ha mencionado varias veces ya que el problema se ha dividido en módulos pero no se ha explicado el proceso mediante el cual se llegó a ese diseño final ni las razones para hacerlo de esa forma. Los siguientes subapartados se estructuran de forma cronológica para ayudar a una mejor comprensión del proceso llevado a cabo.

2.1. Primeros pasos y diseño general

Se consideró de gran importancia entender en profundidad el sistema de TTS (ver arquitectura en el Anexo A) que se iba a emplear para esta tarea, como se detalló en el apartado 1.3. **Coqui**[2], el framework que se decidió usar, tiene una API que facilita el uso de multitud de modelos de TTS, gracias al trabajo previo de análisis del estado del arte llevado a cabo por el departamento ya sabíamos que el mejor modelo para esta tarea era el **Xtts v2**[3]. Lo que nos llevó a usar este modelo frente a otros fue que se trata de un modelo multilenguaje que soporta hasta 17 idiomas a día de hoy, además tiene otras funciones de gran utilidad como clonado de voz sin necesidad de entrenamiento y la transferencia de estilo entre diferentes idiomas. Esto conseguiría generar voces con parecido similar a una voz de referencia independientemente del idioma de entrada.

Las primeras pruebas se llevaron a cabo en la demo[4] que ofrecía el propio framework para evaluar la calidad de los audios y ver como cambiaba el estilo dependiendo del audio de referencia. También se hizo una comparación en local de los modelos **Xtts v2**[3] y **Tacotron 2** [5], que fue unos de los primeros modelos capaces de producir audio a partir de texto. **Tacotron 2** [5] fue un modelo propuesto por Google en 2017, su antecesor **Tacotron** solo generaba espectrogramas de mel.

Tras testear los modelos tts y apreciar las diferencias de calidad entre ambos se vio que al ser modelos generativos el resultado no era determinista, es decir dependiendo de la ejecución una misma linea de texto podía durar mas o menos, introducir pausas aleatorias y sobre todo variar mucho el tiempo de pausa en los signos de puntuación.

Antes de empezar de trabajar en el resto de módulos se diseño un sistema de TTS al que dado un texto, este generaría el audio final por chunks divididos tomando los signos de puntuación como referencia. Además los chunks generados pasan por un proceso de eliminado de silencio en el que se detectan las partes con silencio en los extremos y se eliminan, gracias a esto nos aseguramos un mayor control sobre las pausas generadas por el modelo generativo como se puede ver en la figura 2.

[2] `https://github.com/coqui-ai/TTS`

[3] `https://huggingface.co/coqui/XTTS-v2`

[4] `https://huggingface.co/spaces/coqui/xtts`

[5] Paper: `https://arxiv.org/pdf/1712.05884`

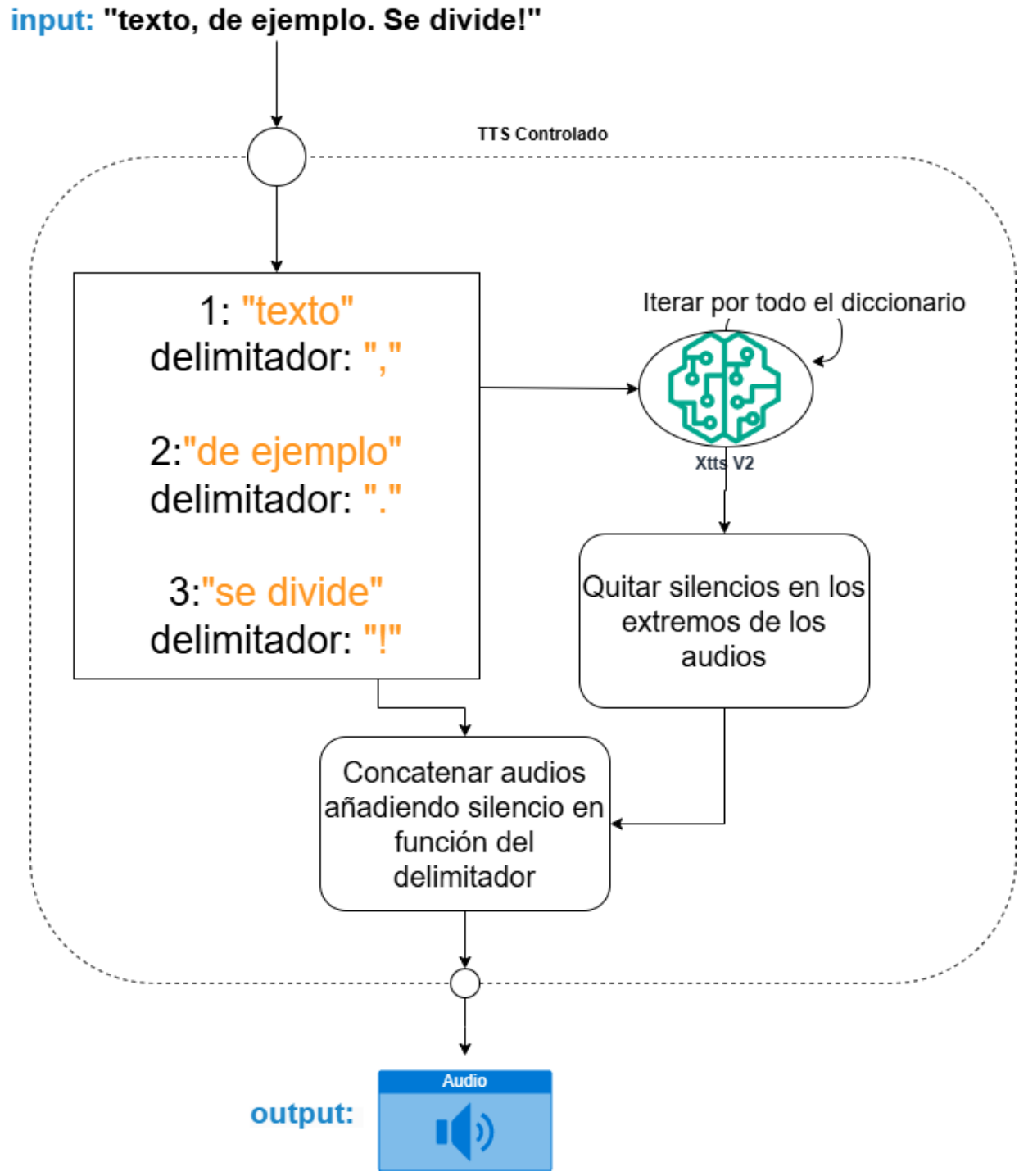

Figura 2: Representación del sistema de TTS controlado

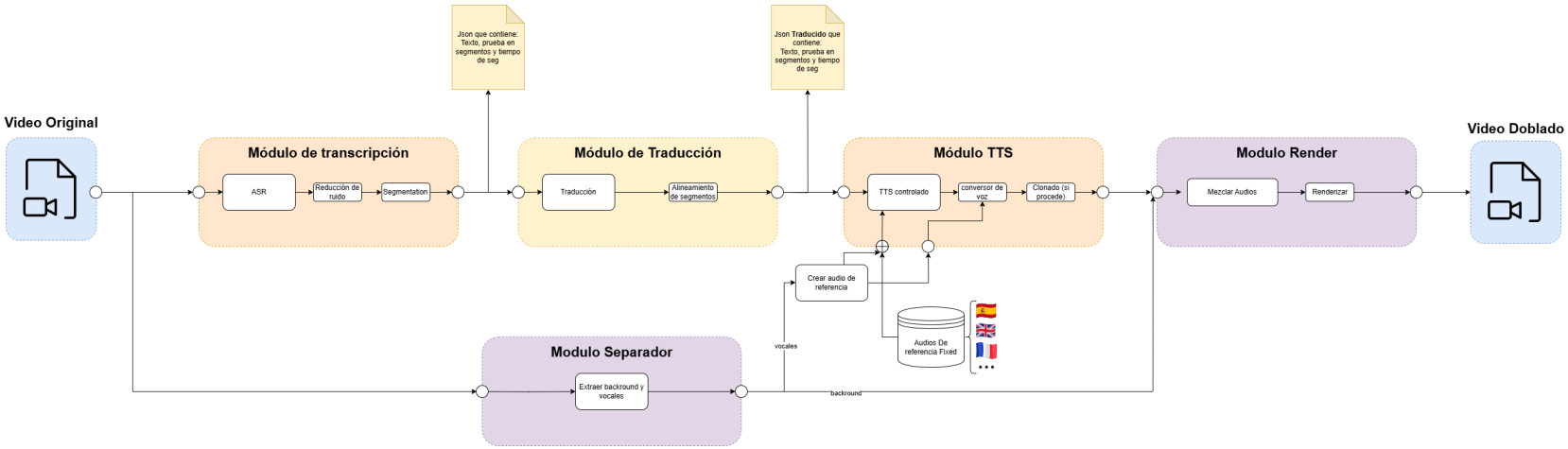

Figura 3: Diagrama de la arquitectura general del programa de doblaje diseñado

Una vez se tuvo clara la parte más básica del TTS se empezó a diseñar la arquitectura general. Como visión general lo que se quería construir era un sistema que dado un vídeo produjera una versión de ese vídeo doblado al idioma seleccionado, como se ve en la figura 3.

La información mas útil para doblar el vídeo se encuentra en la pista de audio, así que lo primero que se hizo fue separar la pista de audio para poder trabajar solo con ella.

Una vez conseguido el audio se pasó al **primer módulo**, se requería de un módulo que fuese capaz de obtener la transcripción del audio, pero no nos valía con tener el texto simplemente. Se necesitaba alta precisión para que la sincronización del audio generado fuese perfecta o lo mas precisa posible.

Partiendo de la salida de ese módulo, abstrayéndonos del formato tendríamos texto, texto que necesitaría ser traducido al idioma deseado por el usuario, entraría el módulo de traducción.

Una vez conseguida la traducción solo quedaría generar el nuevo audio en el idioma deseado y volver a juntar el audio con el audio original, construyendo al final el vídeo en el nuevo idioma. A este *pipeline* quedaría añadirle un pequeño módulo de tratamiento de audio en el que se separase el audio de los hablantes y el audio del fondo, para poder usar ese mismo audio en el nuevo vídeo y desechar las vocales del vídeo original.

Con la arquitectura diseñada de la figura 3 se empezó a trabajar en cada módulo por separado, se optó por un diseño modular porque a la hora de debuggear y testear componentes el programa final sería mucho mas robusto, además facilitaría la actualización de partes del código y su uso a futuro ya que si próximamente apareciese un sistema de TTS mejor, un traductor mejor o un transcriptor mejor, reemplazar esos componentes no afectaría al resto del sistema y sería relativamente sencillo. Además permite mejorar el sistema añadiendo nuevos módulos en cualquier parte de *pipeline* o modificando los actuales sin precisar de un conocimiento en profundidad del resto del código.

2.2. Módulo de transcripción

El módulo de transcripción ha sido la base sobre la que operan el resto de módulos. Se empezó por determinar que modelo de transcripción era el mas apropiado para la tarea. Tras comparar soluciones *open source* como *wav2vec2* y *whisper* y discutirlo con el departamento de audio se llegó a la conclusión de que la mejor opción para esta tarea era la librería **whisper**[6] de *openai*[7].

Se decidió usar *Whisper*[6] en lugar de *wav2vec2* debido a su mayor precisión en la transcripción, especialmente en entornos ruidosos y con acentos variados. Además existía una librería que facilitaba en gran medida el trabajo con los distintos modelos de *openai*[7].

Se empezó por diseñar una clase sencilla de nombre "ASR" con un método que fuese capaz de devolver la transcripción de un audio. Una vez se tenía un transcriptor funcional pasamos a obtener la misma salida pero con la diferencia de que también se incluyeron los tiempos/palabra. Tal y como se mencionó en la sección 1 necesitábamos obtener las *frases prosódicas* que usaríamos para delimitar la transcripción en segmentos. Para determinar los límites de cada frase se ha usado un umbral de **300 ms** como en [10] y [9].

[6] `https://openai.com/index/whisper/`

[7] `https://openai.com`

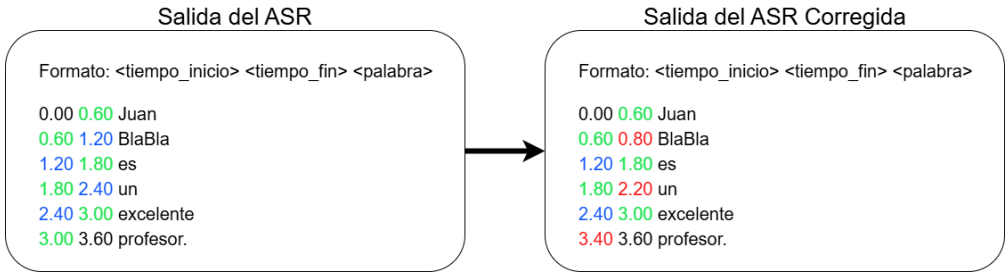

Figura 4: Formato de salida del ASR y ejemplo de corrección

Llegados a este punto se había conseguido una clase capaz de transcribir un texto y devolver un archivo con los tiempos/palabra transcritos. Haciendo uso del umbral mencionado se añadió la lógica de separación en segmentos. Tras una serie de comparaciones para verificar el módulo se apreció que la precisión se los segmentos no era buena, no porque el umbral no fuese el apropiado sino porque el propio transcriptor alargaba los tiempos finales de cada palabra, en muchas ocasiones uniéndolas con el inicio de la siguiente.

Esto creo la necesidad de añadir un post procesado a los tiempos originales obtenidos por el transcriptor y para ello se creo una nueva clase. Esta nueva clase tendría un método principal, el cual devolvería una lista de intervalos con silencios de igual o mayor duración que el *umbral* preestablecido. Tras varios test se ajustaron los valores de los parámetros de *threshold* para obtener los mejores resultados.

La idea era sencilla pero resultó bastante eficaz, detectando silencios en el audio que el transcriptor no había reflejado en la lista de tiempos/palabra, pero aún podía ser mas preciso. Tras dibujar las ondas de audio sobre las que se habían hecho test como se puede apreciar en la figura 5 el ruido en el audio generaba imprecisiones para el detector de silencios.

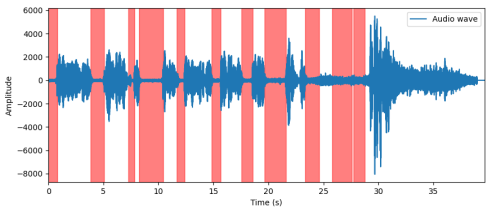

Figura 5: Detección de silencios sin eliminación de ruido

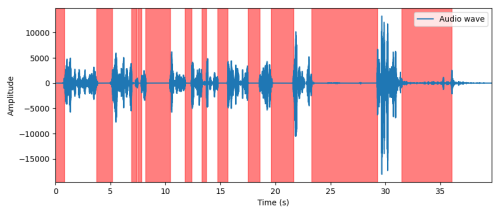

Figura 6: Detección de silencios con eliminación de ruido

Para dar solución a este problema no bastaba con ajustar los parámetos de detector de silencios ya que había ciertas partes del discurso (habla) que estaban al mismo nivel que el ruido presente en cuanto a decibelios. Se exploraron posibles vías y se llego a la solución de usar la librería *noisereduce* [8] que aparecía referenciada en [7]. Se eligió dicha librería por la facilidad de uso que brindaba la API y por la buena compatibilidad entre dependencias del proyecto. La biblioteca de reducción de ruido, como se describe en el documento, utiliza algoritmos avanzados para filtrar componentes irrelevantes de las grabaciones de sonido. Entre las técnicas empleadas,

[8] `https://pypi.org/project/noisereduce/`

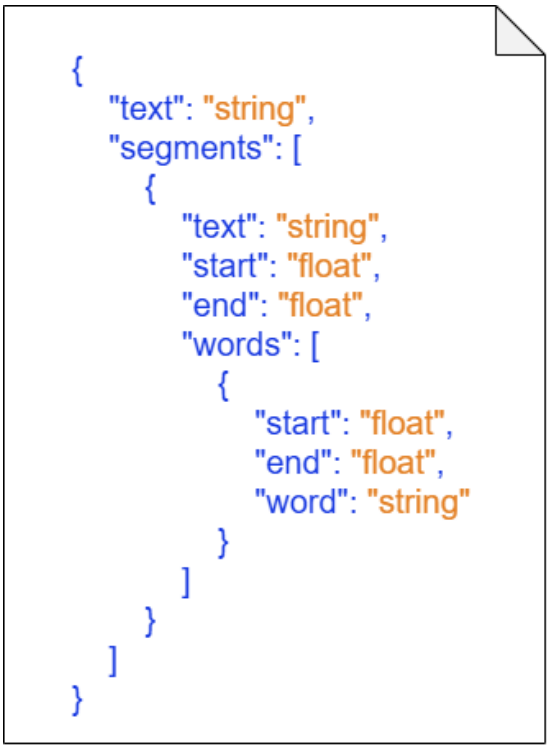

Figura 7: Esquema del archivo de salida de módulo de *Transcripción*

se encuentra el gating espectral, que descarta componentes de tiempo-frecuencia por debajo de un umbral determinado, considerándolos ruido. Además, se utiliza la reducción de ruido no estacionario, que se enfoca en eliminar ruidos de fondo fluctuantes en el tiempo, como el ruido de un avión pasando. La biblioteca implementa estos métodos, permitiendo una estimación dinámica del ruido de fondo y ajustando el umbral de gating en función de la longitud esperada de la señal.

Como se puede apreciar en la figura 6 la detección de los intervalos de silencio resultó mucho mas precisa, lo que consiguió que los segmentos quedasen corregidos en el archivo que generaría el el transcriptor.

Para la salida de la clase *ASR* se decidió estructurar la información en formato *json* [9] con la estructura de la figura 7.

2.3. Módulo de traducción

Partiendo de la salida del módulo anterior tenemos una transcripción precisa dividida en segmentos separados por un umbral de tiempo o también conocido como frases prosódicas. Cada frase prosódica tiene anotada un intervalo de tiempo como se vio en la figura 7.

La tarea de este módulo fue traducir el archivo *json*[9] del módulo de transcripción a otro idioma, manteniendo la delimitación de los segmentos.

[9] JavaScript Object Notation (JSON) es el formato que permite transferir estructuras de datos desde cualquier lenguaje a formatos reconocibles por otros lenguajes y plataformas

2.3.1. Traducción sencilla

Cómo en el módulo anterior la primera aproximación fue crear una clase sencilla que fuese capaz de traducir un texto de un idioma a otro. Para ello fue necesario hacer uso de un modelo pre-entrenado de traducción pero había muchas opciones. En un proyecto anterior a este en la empresa había se realizado un análisis de modelos de traducción, lo que ayudó bastante a elegir el modelo que iba a ser usado. Entre las mejores opciones básicamente hay dos:

- usar un modelo grande (en cuanto a tamaño en memoria y requerimientos de hardware) que sea capaz de traducir múltiples idiomas
- usar modelos de una única dirección de pares de idiomas

Ambos presentan ventajas y desventajas. Los modelos grandes multilenguaje como el *M2M100* [10] cubren muchos pares de idiomas y tienen puntuaciones BLEU bastante buenas ($> 30 - 40$), pero el modelo pesa 5 GB y la inferencia es mas lenta.

Por otra parte existen los modelos *OPUS MT* [11] que son modelos de una única dirección de pares de idiomas. Por ejemplo el modelo ES-EN, traduce texto en español a inglés. Esto hace que sean modelos que pesan mucho menos (300 Mb) y realicen la inferencia mucho mas rápido. Además los modelos *OPUS MT* [11] son capaces e correr en CPU mientras que el resto no.

El grupo de procesamiento del lenguaje natural de la Universidad de Helsinki ha entrenado muchos modelos de traducción utilizando Marian[12] con datos paralelos recopilados en Opus[13] y ha publicado esos modelos como código abierto. Posteriormente, también realizaron una conversión de los modelos entrenados a los Transformers de Huggingface y los pusieron a disposición a través del Huggingface Hub.

MarianMT es una clase en los Transformers de Huggingface para los modelos Marian[12] importados. Se puede entrenar un modelo en Marian [12] y convertirlo a este formato. Los modelos OpusMT son modelos Marian[12] entrenados con los datos de Opus[13] en Helsinki y convertidos a modelos PyTorch. Al buscar en el Huggingface Hub por Marian[12] , se pueden encontrar otros modelos MarianMT además de los provenientes de Helsinki.

Teniendo todo lo mencionado en cuenta se optó por usar los modelos *OPUS MT* [11] mediante la librería Transformers de Huggingface. Se empezó por diseñar una clase *Traductor* que tuviese un método capaz de traducir un texto de un idioma a otro. Al inicializar se pasaría el idioma de entrada y de salida como parámetros y con esos campos se descargaría el modelo pertinente de la forma:

```
1 class Translator:
2     def __init__(self, source_language, target_language="es"):
3         model_name = f"Helsinki-NLP/opus-mt-{source_language}-{
    target_language}"
4         self.model = MarianMTModel.from_pretrained(model_name,
    output_attentions=True)
5         self.tokenizer = MarianTokenizer.from_pretrained(model_name)
```

Listing 1: Elección del modelo de traducción

La clase *Traductor* era capaz de traducir de cualquier idioma a cualquier idioma siempre que el modelo pertinente existiese. Al realizar las pruebas de traducción se vio que la traducción no siempre funcionaba bien con mas de una frase. En ocasiones la salida se generaba sin problema

[10] https://huggingface.co/facebook/m2m100_1.2B

[11] https://github.com/Helsinki-NLP/Opus-MT

[12] Una herramienta de código abierto para el entrenamiento y servicio de traducción automática neuronal

[13] https://opus.nlpl.eu/

y en otras ocasiones la primera frase era la única salida. Por esto se tomó la decisión hacer que el método de traducción dividiese el input de entrada en frases y concatenase el resultado tras la inferencia del modelo, frase a frase y esto solucionó el problema.

2.3.2. Traducción alineada

Una vez conseguida la traducción de un texto de cualquier extensión se empezó a diseñar la clase que atacaría el problema de traducción segmentada. El archivo con la transcripción proporcionado por la clase anterior tiene segmentos cuya longitud es ≥ 1 palabra. Inicialmente se planteó traducir segmento a segmento y generar el archivo traducido, pero de esa forma se perdía el contexto de cada segmento ya que una misma palabra tiene distintas traducciones en otro idioma dependiendo de las palabras que la rodean. Entonces en vez de traducir por segmentos se traduciría por frases, ya que proporcionan una estructura semántica suficiente como para que el modelo tenga suficiente contexto.

Cabe recalcar que la estructura de el archivo de transcripción no está dividida en frases, sino en segmentos[14] los cuales son vitales para una generación de subtítulos sincronizados con el discurso del hablante. El **problema de alineamiento** de segmentos se ha definido como: *dado un texto con segmentos y su versión traducida sin segmentos, segmentar el texto traducido de la forma mas similar posible al texto original.*

A este problema se le da una solución en [5], donde se explica que haciendo uso de la atención[15] se puede ver la relación entre las palabras del texto original y las del texto traducido.

En el *paper* referenciado anteriormente se muestra una matriz en la que se puede apreciar la relación que tienen unas palabras con otras como en la figura 11. Se empezó por averiguar que representaba exactamente esa matriz y como se podía obtener una para nuestra clase de traducción.

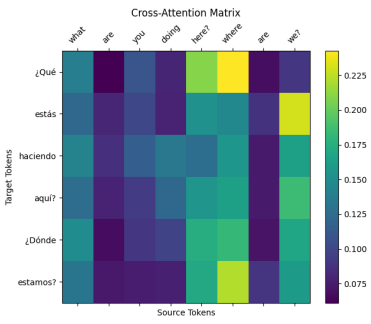

Figura 8: Pesos de atencion cruzada

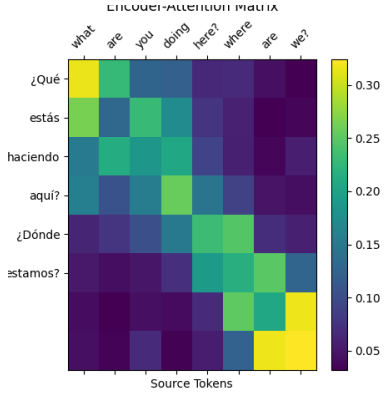

Figura 9: Pesos de atención del encoder

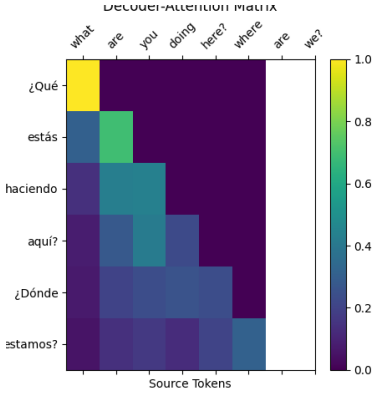

Figura 10: Pesos de atención del decoder

Se intentó replicar la matriz haciendo un plot de los valores del *encoder* y el *decoder* como se puede ver en las figuras 9 y 10 respectivamente. Después se juntaron pero no aportaban gran ninguna información relevante y no tenía parecido con la matriz en [5]. Anteriormente se había probado a sacar la matriz de atenciones cruzadas ya que era lo que más sentido tenía

[14] frases prosódicas

[15] permite al modelo centrarse en partes específicas de la entrada durante el proceso de codificación

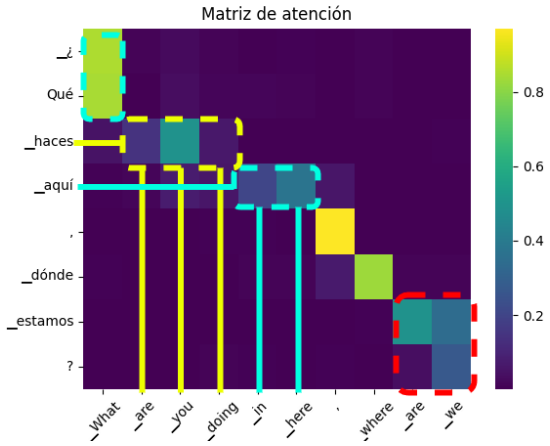

Figura 11: Matriz de atención cruzada con explicación de relaciones

relacionando los valores de entrada y salida, pero se obtuvieron resultados como los que se pueden apreciar en la figura 8.

Llegados a este punto, se optó por contactar a las personas que habían publicado el artículo en cuestión para solicitar ayuda y comprender los errores cometidos. Se descubrió que la utilización de los pesos de atención cruzada era correcta; sin embargo, se había cometido un error en la tokenización de la entrada, al omitir un token adicional específico de la librería. Este token, `<pad>`, era esencial para garantizar la uniformidad en las longitudes de las secuencias de entrada. Esto resulta crucial para el procesamiento eficiente en lotes y la correcta aplicación de máscaras de atención en los modelos Transformers. La ausencia de este token puede provocar errores de dimensión, máscaras de atención incorrectas y problemas de alineación en la secuencia, afectando negativamente el rendimiento del modelo y la calidad de las predicciones. Además, se obtuvo la información de que debía utilizarse la última capa y, en lugar de promediar las cabezas de esta capa, debía usarse solo una de ellas, la que obtuviera los mejores resultados. Esto último fue de gran ayuda ya que no todas las cabezas proporcionaban información relevante y al hacer la media de los valores quedaban matrices que no se parecían tanto al resultado deseado.

Analizando las matrices de atenciones cruzadas de las diferentes cabezas de la última capa se vio que la cabeza que mejor resultado ofrecía era la 4. Algo que no se ha mencionado hasta ahora es que para obtener los pesos de atención no se podía hacer durante la inferencia del modelo, sino que era necesario una vez obtenida la salida, forzar los valores en en *encoder* y *decoder* (véase explicación del transformer en el Anexo A) para obtener los pesos de atención. En cuanto al tiempo empleado no se apreció casi diferencia así que no se le dio mas importancia.

2.3.3. Algoritmo de alineamiento y salida

A partir de la matriz de atención obtenida se decidió diseñar un algoritmo propio para segmentar correctamente la traducción. Para empezar en la matriz obtenida había
(n^o *tokens entrada* $\times$ n^o *tokens salida*) valores pero en realidad muy pocos de todos ellos eran de utilidad.

Dada una matriz de pesos $G \in R^{m \times n}$, se tomó la decisión de construir una matriz binaria $B \in \{0,1\}^{m \times n}$ donde:

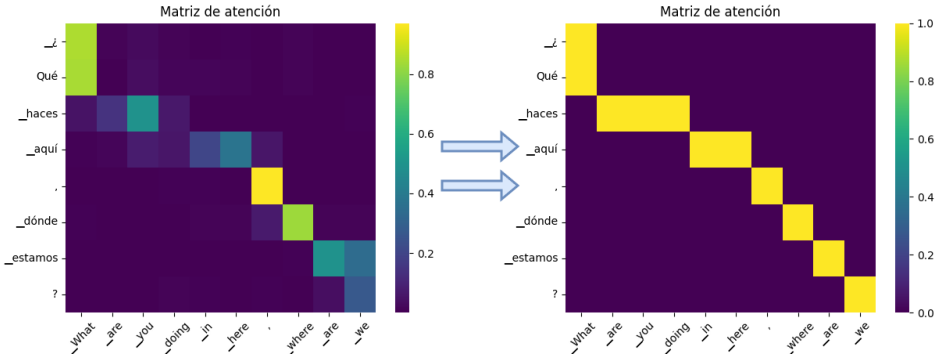

Figura 12: Ejemplo de post-procesado a la matriz de atención

$$b_{ij} = \begin{cases} 1 & \text{si } g_{ij} = \text{máx}(G_{i,\cdot}) \text{ o } g_{ij} = \text{máx}(G_{\cdot,j}) \\ 0 & \text{en caso contrario} \end{cases} \tag{1}$$

Donde: - $G_{i,\cdot}$ representa la fila i de G. - $G_{\cdot,j}$ representa la columna j de G.
De manera más detallada:

$$b_{ij} = \begin{cases} 1 & \text{si } g_{ij} = \text{máx}_k(g_{ik}) \text{ o } g_{ij} = \text{máx}_k(g_{kj}) \\ 0 & \text{en caso contrario} \end{cases} \tag{2}$$

Este procedimiento garantiza que B tendrá unos en las posiciones correspondientes a los valores más altos en cada fila y columna de G.

Por lo tanto, la matriz B guarda como 1 los valores más altos de cada fila y columna, y deja el resto en 0, reflejando así la probabilidad más alta de relación entre dos tokens para cada fila y columna de G. Tras un periodo de pruebas se aprecio lo siguiente:

1. en ocasiones presencia de espurios en la matriz (alucinaciones del modelo)
2. el cambio del orden de un idioma a otro hace que la matriz no refleje la relación que se busca

Para mitigar los efectos negativos que esto provocaría en un algoritmo dependiente de la matriz como única referencia para segmentar el texto, se diseñó un algoritmo que colocaría los separadores en el texto traducido siguiendo las relaciones de la matriz proporcionada siempre que fuese posible y aproximando los separadores que fuesen oportunos siguiendo el flujo del algoritmo 2

Para encontrar la posición del token correspondiente en la traducción donde colocar el separador se implementó el método `_find_index_given_a_row` que buscaba determinar la fila y columna de la matriz binaria donde se encontraba el valor 1, con la restricción de que la columna no excediese un límite dado como se puede ver en el algoritmo 1.

Como se puede ver en el algoritmo 3 una vez conseguida la segmentación la cadena pasaría por un post procesado final para convertir los tokens al texto que se usaría en el módulo de TTS y esto fue necesario porque en ocasiones un separador podía haber acabado entre dos tokens que formaban una palabra por ejemplo. También cabía la posibilidad de que un segmento quedase vacío, asi que de ser así la frase afectada pasaría por una reponderación de los segmentos aproximados teniendo en cuenta el numero de palabras entre cada segmento en la frase original y el número de palabras de la frase traducida.

Algorithm 1 Encontrar la posición del token correspondiente

```
1: procedure FindIndexGivenARow(matrix, collum)
2:     collum ← collum + 1
3:     highest_row ← 0
4:     collum_of_highest_row ← 0
5:     for j = 0 to collum do
6:         for i = 0 to number of rows in matrix do
7:             if matrix[i][j] == 1 then
8:                 if i ≥ highest_row then
9:                     highest_row ← i
10:                    collum_of_highest_row ← j
11:                end if
12:            end if
13:        end for
14:    end for
15:    return highest_row, collum_of_highest_row
16: end procedure
```

Algorithm 2 Colocar Segmentos en el Texto de Destino

```
1: procedure PlaceSegmentsInTargetText(seps, source_words, target_words, matrix)
2:     for cada sep en seps do
3:         if el segmento está al principio del texto de origen then
4:             Añadir delimitador al texto de destino
5:         else if el segmento está al final del texto de origen then
6:             Añadir el resto de palabras de destino y un delimitador al texto de destino
7:         else
8:             Encontrar la mejor coincidencia en la matriz para el segmento actual (FindIndexGi-
   venARow //Algoritmo 1)
9:             if coincidencia exacta then
10:                Añadir las palabras correspondientes al texto de destino y un delimitador
11:            else
12:                Usar una aproximación para añadir las palabras al texto de destino y un delimi-
   tador
13:            end if
14:        end if
15:    end for
16:    if quedan palabras de destino sin procesar then
17:        Añadir las palabras restantes al texto de destino
18:    end if
19:    if hay segmentos vacíos then
20:        Aproximar la colocación de los segmentos vacíos
21:    end if
22:    return target_text, target_seps
23: end procedure
```

Para terminar se transformó el texto en un *json* [9] guardando la misma estructura que en el modulo de *Transcripción*.

Algorithm 3 Algoritmo General de Traducción

```
1: procedure TRANSLATE
2:     Leer y copiar la estructura del archivo JSON de fuente
3:     Extraer texto del JSON y añadir tokens delimitadores de segmento
4:     Dividir el texto en frases
5:     for cada batch de frases do
6:         Obtener el batch actual de frases
7:         Contar delimitadores en el batch
8:         if hay delimitadores then
9:             Traducir el batch de frases
10:            PlaceSegmentsInTargetText //Algoritmo 2
11:            Agregar la traducción procesada al texto traducido
12:        else
13:            Traducir el batch sin delimitadores
14:            Agregar la traducción al texto traducido
15:        end if
16:    end for
17:    Reorganizar los segmentos en el texto traducido si procede
18:    Exportar el texto traducido a un nuevo archivo JSON
19: end procedure
```

2.4. Módulo de TTS

Una vez se consiguió tener el texto y los tiempos el siguiente paso era sintetizar el audio que doblaría la voz del hablante original. En la sección 2.1 se ha hablado de los primeros pasos y las pruebas que se llevaron a cabo con el modelo **Xtts v2**[3], ahora tocaba integrar la clase *Synthesither* como módulo del programa principal de doblaje y añadir los métodos necesarios.

2.4.1. Blending

Tal y como se explica en [5] para que el audio generado se ajuste a un segmento de duración determinado se calcula un *blending ratio* [16] que se usará para modificar la duración del audio generado por el TTS.

Se optó por generar los audios segmento a segmento e ir concatenando audio con audio introduciendo el silencio que tocase entre cada segmento.

Tras un periodo de pruebas se observo que para vídeos de corta duración *(30 segundos - 1 minuto)* la sincronización era perfecta, pero conforme aumentaba el tiempo se añadía un offset al audio generado que hacía que el habla acabase totalmente desfasada. En un primer momento se pensó que se debía a una falta de precisión en la función de cambio de longitud del audio. Tras mejorar la función para admitir muchos mas decimales el offset se había reducido, pero seguía habiendo.

[16] (tiempo del segmento / tiempo del audio generado para el segmento)

Resultó que las operaciones de la biblioteca *Pydub* [17] que se usaba para la concatenación del audio no eran lo suficientemente precisas y los silencios generados iban acumulando error.

Para resolver defitinivamente esta imprecisión se realizó un checkeo en cada iteración de la duración total del audio hasta ese momento y la duración ideal añadiendo o quitando silencio en la concatenación. Quitar o añadir milésimas de segundos entre segmentos era imperceptible y fue justo lo que se necesitó para que los segmentos no se descuadrasen.

2.4.2. Audio de referencia

Como se mencionó en el apartado 2.1 el modelo **Xtts v2**[3] permite usar un audio de referencia para realizar una clonación sin entrenamiento y transferencia de estilo del habla. Las pruebas iniciales se realizaron con una voz de prueba que nada tenía que ver con el vídeo que se doblaba quedando un resultado aceptable pero sin ningún elemento prosódico de la voz transferido mas allá del ritmo.

Cabe recordar que la síntesis de audio se realiza por segmentos y que por tanto es posible realizar la inferencia de cada segmento con un audio de referencia distinto. Es decir podía usarse la parte del segmento original únicamente para cada segmento, transfiriendo el estilo segmento a segmento.

Esta primera aproximación no dió buen resultado ya que de un segmento a otro la voz podía cambiar de tono o el audio de referencia podía ser demasiado corto en un segmento y muy largo en otro lo que dio lugar a una síntesis poco consistente en la que no se percibía una sola persona hablando sino una distinta en cada segmento.

Para que el sistema se volviese consistente y los audios tuviesen transferencia de estilo se optó por un enfoque global, es decir, el audio de referencia sería el mismo para todos los segmentos usando el audio de las vocales obtenido al principio de la ejecución del programa.

Tras un periodo de pruebas se observaron dos puntos importantes:

- En ocasiones las vocales no eran de la calidad suficiente o incluso se mezclaban con partes que no eran del discurso como aplausos o gritos del público si los había.
- Audios de referencia demasiado largos no necesariamente conseguían un mejor resultado

Se suponía que a mas tiempo de audio mejor y mas precisa sería la síntesis de voz, pero se vió que era al contrario. En realidad el problema no venía de la longitud del audio sino de la calidad del mismo. Resulta que se había pasado por alto que en los audios largos había presencia de aplausos y ruidos del público, los cuales degradaban la calidad del audio sintetizado notablemente.

Para abordar este problema, se implementó una clase especializada en la creación de un audio de referencia. Este audio garantizaría una síntesis consistente, utilizando el archivo de transcripción obtenido del módulo descrito en la sección 2.2.

El funcionamiento de la clase se describe a continuación:

1. **Ordenar segmentos:** Los segmentos de audio se ordenan en una lista de mayor a menor duración. Cada elemento de la lista contiene los tiempos de inicio y fin del segmento correspondiente.
2. **Crear audio de referencia:** Se construye un audio de referencia concatenando partes del audio que contienen vocales, siguiendo el orden de la lista de segmentos previamente ordenados.

[17] `https://pypi.org/project/pydub/`

3. **Ajustar duración:** La concatenación de los segmentos se realiza hasta alcanzar la duración deseada del audio de referencia.

De esta manera, se obtuvo un audio de referencia coherente y de la longitud especificada, asegurando que las vocales, las cuales eran elementos críticos para una síntesis vocal clara y precisa, estuviesen adecuadamente representadas.

Resuelto este problema solo quedaba averiguar que hacer cuando el audio original no disponía de la calidad suficiente. Para ello se formó una colección de audios de referencia (uno por idioma sintetizado) para asegurar que en caso de no poder contar con un audio de referencia vocal de calidad habría una alternativa para asegurar una buena síntesis de voz que aunque no sería fiel a la original seguiría teniendo buena calidad. La mayor parte de estos audios fue extraída de canales como *Lingo Mastery*[18] en *YouTube* [19].

Otra razón para usar audios de referencia fijos en vez de las propias vocales fue que en ocasiones a pesar de una buena calidad de audio el sintetizador añadía acentos no deseados. En el caso concreto del español en ocasiones derivaba a acentos latinos sin buscarlo.

2.4.3. Conversión de voz

Al usar audio de referencia fijos se perdía la transferencia del estilo del hablante original al hablante doblado.

Para asegurar la transferencia de estilo de hablantes se usó un conversor de voz (véase explicación de la arquitectura en el Anexo A). El modelo que se recomendó usar desde el departamento de audio de la empresa fue *OpenVoice*[20] Con él se creó un módulo de conversión que usaría como referencia el audio creado por la clase descrita en el apartado 2.4.2 y modificaría la salida generada por el TTS.

Para llevar a cabo la conversión, como en [11], se siguieron los siguientes pasos :

1. **Extracción de embeddings:** Se extrajeron los *embeddings*[21] de cada audio.
2. **Carga del modelo:** Se cargó el *checkpoint* del modelo de conversión de voz pre-entrenado.
3. **Inferencia:** Se realizó la inferencia utilizando el modelo cargado.

Por motivos de incompatibilidad de dependencias este módulo se ejecutó en un entorno virtual aparte.

La conversión ayudó a transferir características del hablante original sin introducir acentos extraños cuando no se podía usar el audio de referencia original.

2.4.4. Clonado de voz

A la hora de transferir características vocales de un hablante desconocido y de forma automática tanto la conversión de voz en el modelo **Xtts v2**[3] como el conversor de voz mencionado en el apartado 2.4.3 realizan muy buen trabajo. Pero no se ha mencionado la opción de clonado ya que requiere una fase de entrenamiento de un modelo y no se puede realizar de forma automática a no ser que se conozca el hablante al cual se va a doblar.

[18] `https://www.lingomastery.com/`

[19] `https://www.youtube.com/`

[20] `https://github.com/myshell-ai/OpenVoice`

[21] Una técnica de procesamiento de lenguaje natural que convierte el lenguaje humano en vectores matemáticos

La clonación consiste en entrenar un modelo con aproximadamente 10 minutos de audio de habla de una persona. Una vez entrenado el modelo este actúa de forma muy similar a un conversor de voz, con la diferencia de que no hace uso de audio de referencia. Los resultados que se obtienen con el clonado son muy superiores en cuanto a calidad y similitud con el hablante original.

Usando la librería *so-vits-svc-fork*[22] se podía realizar inferencia con la voz de un modelo preentrenado o entrenar uno propio.

Se optó por entrenar un modelo con la voz del conocido magnate Elon Musk usando como fuente un podcast[23] de Joe Rogan del cual se extrageron segmentos en los que solo hablaba Elon Musk hasta tener aproximadamente 10 minutos de audio.

El entrenamiento duró 15 horas en una RTX 4070 con 12 GB y los resultados de la inferencia fueron muy satisfactorios. De esta forma al como extensión se dió la opción de usar clonado para el audio final o no.

Para suavizar las abruptas transiciones entre habla y silencio generadas por el clonador, se decidió pasar el audio generado por el mismo separador de audio usado al principio del *pipeline* y esto resolvió el problema.

2.5. Programa principal y renderizado

Para finalizar se implementó un módulo de renderizado, el cual tendría como tarea unir el audio de las vocales generado por el módulo de TTS y el audio original de fondo y renderizar el vídeo doblado haciendo uso de la librería *moviepy*[24].

Para facilitar el uso del programa y la realización de pruebas se implementó también un programa main al cual se le especificarían los idiomas de salida y el archivo de video de entrada, además del nombre del proyecto. El programa generaría carpetas en el subdirectorio llamado *resultados* con el nombre de la prueba y dentro de este tantas subcarpetas como idiomas especificados.

En cada una de esas subcarpetas se encontrarían las versiones dobladas del vídeo original en sus respectivos idiomas como muestra la figura 13.

Por temas de compatibilidad entre librerías y dependencias necesarias la se decidió usar conda para crear entornos virtuales, agrupando la mayor parte del código posible en cada entorno.

[22] `https://github.com/voicepaw/so-vits-svc-fork`

[23] `https://youtu.be/ycPr5-27vSI`

[24] `https://pypi.org/project/moviepy/`

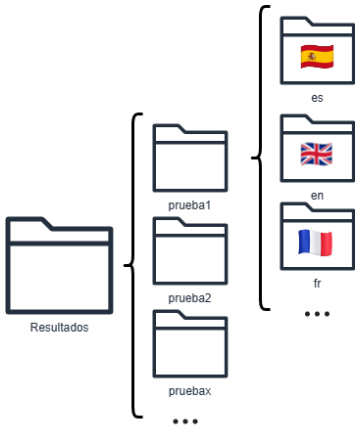

Figura 13: Distribución del árbol de carpetas de los resultados y distintas pruebas

Para gestionar la ejecución de código en diferentes entornos virtuales se implementó una función con el siguiente funcionamiento:

1. **Activar entorno virtual:** dado un nombre pasado como parámetro se activaría el entrono con dicho nombre.
2. **Ejecutar Script:** se ejecutaría el script correspondiente mediante un subproceso
3. **Desactivar entorno virtual:** desactivaría el entorno actual volviendo al entorno previo.

Para asegurar la portabilidad del código se creó una carpeta con tantos archivos como entornos virtuales se usaron, con los requerimientos de cada entorno. Ejecutando el comando `conda env create -f <nombre_archivo>.yml` se replicaría el entorno virtual.

3. Resultados

Cada uno de los módulos se ha ido evaluando conforme se implementaron. Se ha hecho uso de tanto evaluación cualitativa como cuantitativa evaluando cada módulo de forma independiente y el doblaje en general en su conjunto.

3.1. Módulo de transcripción

Para realizar la transcripción en el modulo de la sección 2.2 se usó el modelo de *whisper medium*. Se eligió ese modelo porque dado el requerimiento de VRAM que se puede observar en la tabla 1 este era el modelo más grande que se podía ejecutar en el hardware del que se disponía.

Modelo	**VRAM Requerida**	**Velocidad relativa**
tiny	∼1 GB	∼32x
base	∼1 GB	∼16x
small	∼2 GB	∼6x
medium	∼5 GB	∼2x
large	∼10 GB	1x

Cuadro 1: Comparación de modelos de whisper y sus características.

El modelo ha sido puesto a prueba con distintos *datasets* por la empresa *openai*[7] haciendo uso de la métrica WER (véase fórmula y explicación de la métrica en el Anexo A) para distintos idiomas.

Los resultados de la tabla muestran una gran precisión con los distintos *datasets* y comprando con otros modelos de igual o mayor tamaño, las métricas del modelo *medium* resultaron satisfactorias como se puede ver en las tablas 2 y 3 obteniendo mejores métricas que el resto de modelos fuera de *whisper*.

Modelo	**TED-LIUM3**	**Meanwhile**	**Kincaid46**	**Rev16**	**Earnings21**
Whisper medium	3.5	5.4	8.6	11.0	10.3
wav2vec2-base-100h	17.6	27.7	39.3	35.2	45.7
wav2vec2-base-960h	12.8	19.7	32.9	29.8	37.3
wav2vec2-large-960h-lv60-self	7.2	11.4	21.1	21.3	21.7
wav2vec2-large-960h	10.1	16.4	27.4	26.4	30.4
wav2vec2-large-robust-ft-libri-960h	8.8	15.2	22.9	23.4	23.7
hubert-large-ls960-ft	8.1	12.9	22.4	23.4	23.0
hubert-xlrlage-ls960-ft	8.1	15.7	19.7	24.2	25.1
stt_en_conformer_ctc_large	4.0	9.8	13.1	14.5	12.6
stt_en_conformer_transducer_xlarge	5.3	10.6	17.1	19.8	16.2

Cuadro 2: WER (%) con datasets en inglés de [6]

Modelo	Español	Francés	Inglés	Alemán	Italiano
Whisper medium	9.6	12.2	7.6	12.4	19.7

Cuadro 3: WER (%) con el dataset multilenguaje VoxPopuli de [6]

3.2. Módulo de traducción

Para evaluar el módulo de traducción se recopilaron las métricas BLEU (véase fórmula y explicación de la métrica en el Anexo A) para las direcciones de traducción usadas de la web OPUS[25] donde se pueden observar los resultados de los modelos *OPUS MT* [11] vs modelos externos como el *M2M100*[10].

Dirección de traducción	**Mejor score OPUS MT**	**Mejor score externo**
en-es	57.2	56.2
es-en	62.3	62.7
en-fr	55.7	52.3
fr-en	59.8	60.6
en-it	53.9	54.3
it-en	72.4	71.3
en-du	55.7	57.6
du-en	52.8	49.8

Cuadro 4: Comparación de scores BLEU de traducción

Como se puede ver en la tabla 4 las métricas son bastante parecidas y como se puede ver en el anexo en la tabla 7 por encima de 30 tenemos una buena traducción. Esto unido a la mayor velocidad de inferencia de los modelos *OPUS MT* [11] de la que se habló en la sección 2.3 y el menor tamaño tanto en VRAM como en disco reforzó en la decisión de usarlos.

Cabe mencionar que la métrica BLEU es representativa para evaluar la calidad de las traducciones. Sin embargo, dada su naturaleza y la del problema, es un tanto limitada y debe interpretarse con cautela.

Con respecto a la segmentación realizada en el módulo fue evaluada de forma cualitativa comparando el archivo de transcripción y el de traducción. También se observaron las matrices de atención obtenidas sobre todo analizando los casos en los que había habido alguna imprecisión y se observó que la mayoría de veces se debía al distinto orden de cada idioma o a una alucinación del modelo como se puede ver en la figura 14 Los resultados podían tener imprecisiones de aproximadamente una palabra como mucho así que se dio por buena.

[25] `https://opus.nlpl.eu/dashboard/`

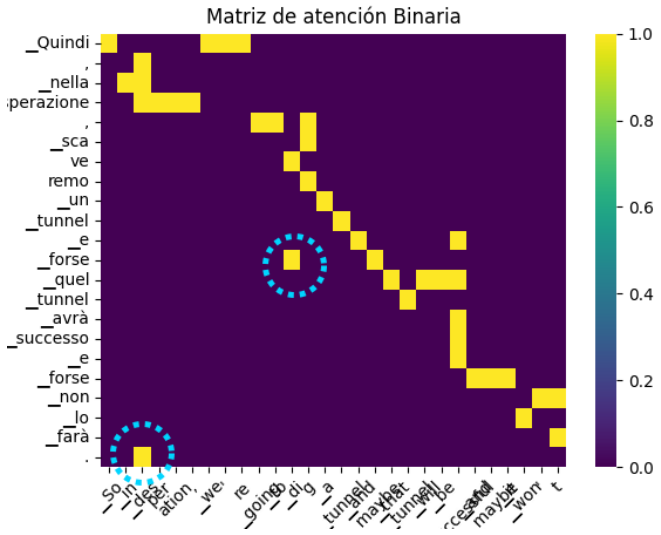

Figura 14: Ejemplo de mala matriz de pesos que requiere aproximación

Cuando una matriz presenta outliers como los que se aprecian en la figura 14 y algún segmento queda vacío, se realiza una aproximación de los segmentos teniendo en cuenta el número de palabras en el idioma original y en el traducido como se menciona en la sección 2.3.3.

Este post-proceso resultó muy útil recuperando segmentos que no se habrían obtenido con la matriz aunque con menos precisión en cuanto a las palabras en cada segmento.

Como la segmentación se pondera en cada frase no nos arriesgamos demasiado, ya que si por el contrario se hiciese con todo el texto, un *outlier* como el de la figura 14 forzaría a aproximar la mayoría de segmentos y añadiría mucho error.

3.3. Módulo de TTS

La parte mas importante del módulo de TTS es conseguir que la voz resultante se entienda y suene lo más parecido posible a una voz humana. Para evaluar el rendimiento de este módulo se han realizado diferentes pruebas.

Se implementó un módulo de evaluación midiendo la puntuación WER del audio generado para distintas versiones del audio final:

- Salida del modelo TTS sin modificar
- Con conversor de voz
- Con clonado de voz

Para implementar dicho módulo se hizo uso de un *script* de código abierto elaborado por el usuario *duckyngo* [26]. De dicho *script* se extrajo una función que calcularía el WER dadas dos cadenas de texto que se pasarían como parámetro.

Esta prueba se realizó para determinar si la salida del sistema era entendible pese a los cambios de velocidad y cortes realizados por el *blending* en el módulo de TTS.

Las puntuaciones WER obtenidas en la tabla 5 representan la diferencia entre el texto reconocido del audio testeado por el modelo de transcripción *whiper medium* y el testo obtenido por

[26] `https://github.com/duckyngo/Word-Error-Rate-Visualization-with-Colab/blob/main/README.md`

Tipo	**Lenguaje**			
	Español	**Francés**	**Italiano**	**Árabe**
tts	0.03	0.15	0.10	0.14
convertido	0.05	0.22	0.10	0.23
clonado	0.10	0.49	0.27	0.63

Cuadro 5: WER por tipo de audio final y lenguaje

el módulo de Traducción descrito en la seccion 2.3. Se optó por usar el módulo de Transcrición como *ground truth* dados sus altos niveles de precisión.

En la tabla 5 se observa que los valores de WER para el audio del modelo TTS y el audio convertido son bastante similares, esto indica que la conversión apenas añade error. Hay que tener en cuenta que el modelo ASR funciona mejor con el español que con el resto como se vio en la tabla 3. En cuando al audio clonado podemos apreciar valores notablemente altos de WER para los idiomas Árabe y Francés. En realidad el valor del audio en Francés refleja la misma progresión que en Español y el Italiano. Por otro lado el audio clonado en Árabe tiene mas error porque es un lenguaje que se aleja mucho del ingles en cuanto a fonemas y a la hora de convertirlo con un modelo que solo se ha entrenado con Inglés el acento resultante hace que se pierda información

Por lo tanto un modelo de clonado podría ser entrenado con frases en distintos idiomas, lo cual podría resultar difícil de conseguir. La otra opción es elegir la versión de conversión ya que apenas añade error y sigue habiendo transferencia de estilo. Para idiomas que tienen un parecido mayor se elegirá el clonado siempre que sea posible.

Para evaluar la calidad del audio generado en cuanto a la similitud con una voz humana se realizó una encuesta con **53 participantes** en la que se mostraban distintas muestras de audio con clonado, con conversión y salidas directamente del modelo TTS para ver que porcentaje de la muestra de encuestados tomaba cada una de las muestras como real.

En la tabla 6 se han detallado los porcentajes de encuestados que han considerado los audios como "reales", es decir, no generados por una máquina. *Audio 1* es un audio generado por el modelo *Xtts v2*[3], al igual que *Audio 2*. La única diferencia es el audio de referencia que se usa.

El audio se generó en inglés. *Audio 1* tiene el audio de referencia en inglés, mientras que *Audio 2* tiene el audio de referencia en español. Esto se hizo para evaluar cómo la credibilidad del audio aumentaba con el uso de conversión y clonado de voz, a pesar de contar con un audio de referencia en otro idioma.

Los resultados resultaron satisfactorios ya que, en promedio, cada audio tenía un 60 % de credibilidad. A continuación, se desglosan los porcentajes específicos para cada combinación de tipo de audio y método de generación:

Tipo de audio	**Porcentaje de encuestados con respuesta "real"**
Audio 2 + clonado	48 %
Audio 1 + clonado	89 %
Audio 2 + conversión	65 %
Audio 1 + conversión	60 %
Audio 2 (audio de referencia fijo, en otro idioma)	40 %
Audio 1	80 %

Cuadro 6: Porcentaje de encuestados que consideran "reales"los audios

Como se puede observar en la tabla 6, los audios generados con la técnica de clonado tienden a ser percibidos como más reales, especialmente cuando se utiliza un audio de referencia en el mismo idioma. En particular, *Audio 1 + clonado* alcanzó un 89 % de credibilidad, el valor más alto registrado. Por otro lado, el uso de un audio de referencia en otro idioma (*Audio 2*) muestra que, aunque la credibilidad disminuye, técnicas como la conversión y el clonado pueden mitigar este efecto negativo. La credibilidad de *Audio 2* mejoró significativamente al utilizar estas técnicas, pasando de un 40 % a un 65 % y 48 % respectivamente.

En conclusión, estos resultados sugieren que las técnicas de conversión y clonado de voz son efectivas para mejorar la percepción de autenticidad de los audios generados, incluso cuando el audio de referencia no está en el mismo idioma que el audio final. Esto abre nuevas posibilidades para la generación de audios creíbles en múltiples idiomas, facilitando aplicaciones en doblaje y otras áreas donde la autenticidad del audio es crucial.

3.4. Resultados generales

Evaluando el doblaje de forma general, se apreciaron buenos resultados en la mayoría de los casos. Los avances en las técnicas de conversión y clonado de voz contribuyeron significativamente a estos resultados positivos, especialmente en los casos en los que el audio de referencia no era el del vídeo original.

Sin embargo, se observaron algunas áreas de mejora. En ocasiones, el proceso de *blending* tiende a acelerar demasiado la velocidad del habla. Esta aceleración excesiva puede afectar negativamente la naturalidad y la inteligibilidad del doblaje, haciendo que el discurso suene forzado o artificial. Actualmente, este problema no puede ser corregido fácilmente sin recurrir a un *rephrase* con un modelo de lenguaje grande (LLM, por sus siglas en inglés).

En resumen, aunque los resultados generales son prometedores y la tecnología de doblaje automático ha avanzado considerablemente, se requiere un enfoque continuo en la optimización de ciertos aspectos técnicos, como el blending, para alcanzar una calidad de doblaje aún más alta y más natural.

3.4.1. Tiempo de ejecución

En la Figura 15, se presenta un gráfico de barras que muestra el porcentaje de tiempo que requiere cada tarea con respecto a la duración del vídeo original. Este gráfico es fundamental para entender cómo se distribuye el tiempo de procesamiento entre las diferentes actividades del *pipeline* de doblaje. (ver las características de la máquina en el Anexo A)

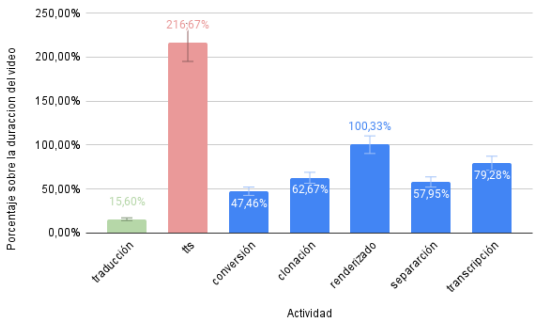

Figura 15: Porcentaje de tiempo que requiere cada tarea con respecto a la duración del video original

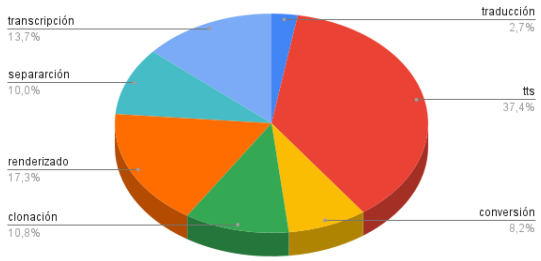

Figura 16: Porcentaje de tiempo que usa cada tarea en todo el *pipeline*

Para ilustrar mejor estos datos, se considera un ejemplo con un video de 30 segundos. A continuación se detalla el tiempo que tardaría cada módulo en procesar este video:

- **Separación:** 30 segundos $\times$ 57.95 % = 17.39 segundos
- **Transcripción:** 30 segundos $\times$ 79.28 % = 23.78 segundos
- **Traducción:** 30 segundos $\times$ 15.60 % = 4.68 segundos
- **TTS:** 30 segundos $\times$ 216.67 % = 65 segundos
- **Conversión:** 30 segundos $\times$ 47.46 % = 14.24 segundos
- **Clonación:** 30 segundos $\times$ 62.67 % = 18.80 segundos
- **Renderizado:** 30 segundos $\times$ 100.33 % = 30.10 segundos

Sumando todos estos tiempos, obtenemos el tiempo total de procesamiento:

- **Tiempo total:** 4.68 + 65 + 14.24 + 18.80 + 30.10 + 17.39 + 23.78 = 173.99 segundos (factor de $5,8$)

En la Figura 16, se presenta un gráfico circular que representa los porcentajes de tiempo que utiliza cada tarea en todo el *pipeline* de doblaje. Este gráfico permite visualizar la distribución del tiempo de procesamiento en una perspectiva global.

De acuerdo con los datos presentados, la tarea de TTS es la más demandante en términos de tiempo, ocupando el 37.4 % del tiempo total del *pipeline*. La traducción, por otro lado, representa solo el 2.7 % del tiempo total, mientras que el renderizado ocupa un 17.3 %. La conversión y la clonación también representan una parte significativa del tiempo total, con un 8.2 % y 10.8 % respectivamente.

Es importante señalar que si se dispone de una tarjeta gráfica con soporte CUDA, todos los módulos excepto el de renderizado incrementarían notablemente su velocidad. Esto se debe a la capacidad de procesamiento paralelo que ofrece CUDA, lo cual es particularmente beneficioso para las tareas de redes neuronales y otras operaciones intensivas en cómputo. De esta manera, la eficiencia del *pipeline* de doblaje podría mejorar significativamente, reduciendo los tiempos de procesamiento y aumentando la capacidad de manejar volúmenes mayores de trabajo en menor tiempo.

En conclusión, la evaluación del tiempo de ejecución revela áreas críticas donde la optimización y el uso de hardware acelerado pueden llevar a mejoras sustanciales en la eficiencia del sistema de doblaje automático.

4. Conclusiones

Una vez terminado el trabajo se ha reflexionado sobre las aportaciones de este, el cumplimiento de los objetivos propuestos inicialmente en la sección 1.2 y las posibilidades de continuación futuras.

4.1. Aportaciones

Como se comentó en la sección 1.1, este trabajo ha sido desarrollado en empresa. Gracias a este trabajo se dispone de un sistema de doblaje que tiene en cuenta los elementos prosódicos del hablante original.

En aproximaciones anteriores de la empresa se realizaron prototipos mejorables en cuanto a la segmentación y prosodia en general. Este trabajo ha servido para mejorar el sistema de doblaje existente añadiendo una segmentación completamente distinta a la que existía previamente, añadiendo un control extra sobre el modelo de TTS para reducir la impredecibilidad del modelo y creando una voz final muy parecida a la humana, manteniendo en la medida de lo posible las características vocales originales.

En general este trabajo supone una mejora con respecto a las arquitecturas de doblaje existentes mezclando diferentes aproximaciones.

4.2. Cumplimiento de los objetivos iniciales

Se puede concluir que todos los objetivos principales han sido alcanzados satisfactoriamente:

1. **Conseguir una buena sincronización del audio doblado sobre el vídeo original**: El módulo de *Traducción* ha logrado una sincronización precisa del audio doblado con el vídeo original. Gracias a las mejoras en el algoritmo de segmentación y la implementación de técnicas avanzadas de alineación, se ha alcanzado un nivel de sincronización que cumple con los estándares esperados, logrando una experiencia de visualización natural y coherente.
2. **Generar una voz que pueda ser interpretada como humana**: Se ha desarrollado un modelo de síntesis de voz que produce una salida con características prosódicas muy similares a las de una voz humana real. La evaluación subjetiva realizada con usuarios finales ha demostrado que la mayoría percibe la voz generada como natural y convincente, cumpliendo así con este objetivo.
3. **Generar doblaje en más de un idioma**: El sistema ha sido diseñado con capacidades multilingües, permitiendo la generación de doblaje en varios idiomas. Se ha implementado una clase de traducción que soporta mas de 100 idiomas y se ha hecho uso de un modelo de TTS multilenguaje, lo que permite al sistema manejar eficazmente el doblaje automático en distintos contextos lingüísticos sin pérdida de calidad en la síntesis de voz.
4. **Diseñar un sistema modular que sea sencillo de mantener y actualizar**: La arquitectura del sistema se ha diseñado de manera modular, facilitando tanto su mantenimiento como futuras actualizaciones. Cada módulo, incluyendo la traducción, la síntesis de voz, la separacion de audio y el módulo de renderizado, puede ser actualizado de manera independiente, lo que permite una adaptación rápida a nuevas tecnologías y métodos sin necesidad de reestructurar todo el sistema.

En resumen, el trabajo realizado ha cumplido con éxito los objetivos iniciales, ofreciendo una solución avanzada y eficaz para el doblaje automático con características prosódicas humanas y capacidades multilingües. Además, el diseño modular asegura la sostenibilidad y escalabilidad del sistema a largo plazo.

4.3. Posibilidades de continuación

En cuanto a posibilidades de continuación se refiere, hay muchas líneas de trabajo posibles:

- **Uso de modelos de lenguaje (LLM) para refrasear segmentos demasiado cortos o largos en la traducción**: Implementar modelos de lenguaje avanzados para mejorar la coherencia y naturalidad de las traducciones. Esto permitiría ajustar la longitud de los segmentos traducidos para que se adapten mejor al ritmo del vídeo original.
- **Segmentación de hablantes para traducir vídeos con varios hablantes**: Desarrollar algoritmos de segmentación de hablantes que permitan identificar y separar las voces de diferentes personas en un vídeo. Esto facilitaría la traducción y doblaje de vídeos con múltiples interlocutores, mejorando la precisión y calidad del doblaje en contextos más complejos que los monólogos.
- **Ejecución en GPU con CUDA y cambio de velocidad**: Aprovechar las capacidades de procesamiento paralelo de las GPUs mediante CUDA para acelerar significativamente la síntesis de voz y otros procesos computacionales intensivos. Esto requeriría disponer del hardware adecuado, pero permitiría una ejecución más rápida y eficiente del sistema.
- **Interfaz gráfica para la edición y corrección de errores**: Desarrollar una interfaz de usuario que permita a los operadores humanos revisar y corregir errores en la transcripción, traducción o síntesis de voz. Aunque el sistema automático no será perfecto, la posibilidad de realizar correcciones humanas sencillas puede elevar considerablemente la calidad del producto final.
- **Sincronizado de labios**: Integrar técnicas avanzadas de sincronización labial para que la voz doblada coincida con los movimientos labiales de los personajes en el vídeo. Esto mejoraría la inmersión y la naturalidad del doblaje, proporcionando una experiencia visual más convincente y coherente.

5. Bibliografía

Referencias

[1] Ketan Doshi. "Foundations of NLP Explained — BLEU Score and WER Metrics". En: (2021). URL: https://towardsdatascience.com/foundations-of-nlp-explained-bleu-score-and-wer-metrics-1a5ba06d812b.

[2] Ketan Doshi. "Transformers Explained Visually (Part 3): Multi-head Attention, deep dive". En: (2021). URL: https://towardsdatascience.com/transformers-explained-visually-part-3-multi-head-attention-deep-dive-1c1ff1024853.

[3] Alon Lavie. "Evaluating the Output of Machine Translation Systems". En: (2011). URL: https://www.cs.cmu.edu/%7Ealavie/Presentations/MT-Evaluation-MT-Summit-Tutorial-19Sep11.pdf.

[4] Dongchao Li et al. "XTTS: Zero-Shot Text-to-Speech Synthesis with a Unified Representation". En: (2024). URL: https://arxiv.org/pdf/2406.04904.

[5] Alp Öktem, Mireia Farrús y Antonio Bonafonte. "Prosodic Phrase Alignment for Machine Dubbing". En: *Proc. IberSPEECH 2019*. Graz, Austria, 2019. URL: https://arxiv.org/pdf/1908.07226.

[6] Alec Radford et al. "Robust Speech Recognition via Large-Scale Weak Supervision". En: (2022). URL: https://arxiv.org/pdf/2212.04356.

[7] Tim Sainburg y Timothy Q. Gentner. "Toward a Computational Neuroethology of Vocal Communication: From Bioacoustics to Neurophysiology, Emerging Tools and Future Directions". En: (2021). URL: https://ieeexplore.ieee.org/document/9414966.

[8] Stefan. "Understanding Transformers and Attention". En: (2023). URL: https://medium.com/@stefanbschneider/understanding-attention-and-transformers-d84b016cd352.

[9] Yogesh Virkar et al. "IMPROVEMENTS TO PROSODIC ALIGNMENT FOR AUTOMATIC DUBBING". En: (2021). URL: https://ieeexplore.ieee.org/document/9414966.

[10] Yogesh Virkar et al. "PROSODIC ALIGNMENT FOR OFF-SCREEN AUTOMATIC DUBBING". En: (2022). URL: https://arxiv.org/pdf/2204.02530.

[11] Tomasz Walczyna y Zbigniew Piotrowski. "Overview of Voice Conversion Methods Based on Deep Learning". En: (2017). URL: https://www.mdpi.com/2076-3417/13/5/3100.

[12] Tomasz Walczyna y Zbigniew Piotrowski. "Overview of Voice Conversion Methods Based on Deep Learning". En: *Applied Sciences* 13.5 (2023), pág. 3100. URL: https://doi.org/10.3390/app13053100.

A. Anexo I

A.1. Explicación de la puntuación BLEU [1]

La métrica BLEU (Bilingual Evaluation Understudy) se utiliza para evaluar la calidad de texto generado por modelos de traducción automática u otros modelos de generación de texto. Tal y como se detalla en [1], se basa en la comparación de n-gramas del texto generado con los n-gramas de una o más referencias humanas. El cálculo de BLEU involucra:

A.1.1. Tabla BLEU

Puntuación BLEU	Interpretación
< 10	Casi inútil
$10 - 19$	Difícil de capar la esencia
$20 - 29$	La esencia es clara, pero tiene errores gramaticales significativos
$30 - 40$	Comprensible por buenas traducciones
$40 - 50$	Traducciones de alta calidad
$50 - 60$	Traducciones de calidad muy alta, adecuadas y fluidas
> 60	Calidad generalmente mejor que la humana

Cuadro 7: Interpretación de las puntuaciones BLEU a partir de [3]

A.1.2. N-gramas

Un n-grama es una secuencia de n palabras consecutivas en una oración. Por ejemplo, en la frase "The ball is blue":

- Unigramas (1-gram): "The", "ball", "is", "blue".
- Bigramas (2-gram): "The ball", "ball is", "is blue".
- Trigramas (3-gram): "The ball is", "ball is blue".
- 4-grama: "The ball is blue".

A.1.3. Precisión

La precisión se calcula como:

$$\text{Precisión} = \frac{\text{Número de n-gramas correctos}}{\text{Número total de n-gramas predichos}} \tag{3}$$

Para evitar el problema de repetición y manejar múltiples oraciones de referencia, se usa la precisión recortada (clipped precision).

A.1.4. Penalización por brevedad

Para penalizar las oraciones demasiado cortas, se introduce una penalización por brevedad (*Brevity Penalty*, BP):

$$\text{BP} = \begin{cases} 1 & \text{si } c > r \\ e^{(1-\frac{r}{c})} & \text{si } c \leq r \end{cases} \tag{4}$$

donde c es la longitud de la oración generada y r es la longitud de la oración de referencia.

A.1.5. Cálculo del BLEU

Finalmente, el puntaje BLEU se calcula como:

$$\text{BLEU} = \text{BP} \cdot \exp\left(\sum_{n=1}^{N} w_n \log p_n\right) \tag{5}$$

donde p_n es la precisión de los n-gramas y w_n son los pesos que normalmente son iguales para todos los n-gramas.

A.2. Explicación de la puntuación WER

La métrica Word Error Rate (WER) se utiliza para evaluar la precisión de modelos de reconocimiento de voz. Compara la transcripción generada por el modelo con una transcripción de referencia, y se calcula como:

$$\text{WER} = \frac{S + D + I}{N} \tag{6}$$

donde:

- S es el número de sustituciones.
- D es el número de eliminaciones.
- I es el número de inserciones.
- N es el número total de palabras en la transcripción de referencia.

WER proporciona un porcentaje de error, siendo 0 % el valor óptimo (sin errores).

A.3. Arquitectura del Modelo XTTS [4]

El modelo XTTS (Zero-Shot Text-to-Speech multilingüe) está diseñado para generar voz en varios idiomas. Este modelo se basa en el modelo *Tortoise*[27], pero se ha modificado para soportar el entrenamiento en múltiples lenguas, mejorar la clonación de voces y acelerar tanto el entrenamiento como la inferencia. La arquitectura del XTTS consta de tres componentes principales: VQ-VAE, un codificador GPT-2, y un decodificador basado en HiFi-GAN.

A.3.1. Componentes del Modelo

VQ-VAE: El modelo utiliza un Autoencoder Variacional con Cuantización Vectorial (VQ-VAE) con 13 millones de parámetros. Este componente toma como entrada un mel-espectrograma y codifica cada cuadro utilizando un único libro de códigos con 8192 códigos a una tasa de 21.53 Hz. Luego, se filtra este libro de códigos para mantener solo los 1024 códigos más frecuentes, lo cual incrementa la expresividad del modelo.

Codificador GPT-2: El codificador es un transformador basado en GPT-2 que cuenta con 443 millones de parámetros. Este componente toma los tokens de texto, que se obtienen a través de un tokenizador personalizado de Byte-Pair Encoding (BPE) con 6681 tokens, y predice los códigos de audio VQ-VAE. Además, se condiciona con un codificador de condiciones, que procesa los mel-espectrogramas para generar 32 embeddings de 1024 dimensiones por muestra de audio. Este diseño incluye capas de atención escalada de 16 cabezas y un Resampler Perceiver para generar un número fijo de embeddings, mejorando la capacidad del modelo para la clonación de voz en un entorno multilingüe.

Decodificador HiFi-GAN: El decodificador se basa en el vocoder HiFi-GAN y tiene 26 millones de parámetros. Este componente toma los vectores latentes generados por el codificador GPT-2. Debido a la alta tasa de compresión del VQ-VAE, reconstruir el audio directamente desde los códigos VQ-VAE puede causar problemas de pronunciación y artefactos. Para evitar esto, el espacio latente del codificador GPT-2 se usa como entrada para el decodificador. El decodificador también se condiciona con embeddings de hablantes del modelo H/ASP, que se añaden en cada capa de upsampling mediante proyección lineal. Además, para mejorar la similitud con las voces de los hablantes, se incorpora la Pérdida de Consistencia de Hablante (SCL).

[27] `https://docs.coqui.ai/en/latest/models/tortoise.html`

A.4. Explicación del Transformer

El transformer es una arquitectura de red neuronal basada en el mecanismo de atención, presentada en el artículo "Attention is All You Need"por Vaswani. La arquitectura del transformer consta de dos componentes principales: el codificador y el decodificador.

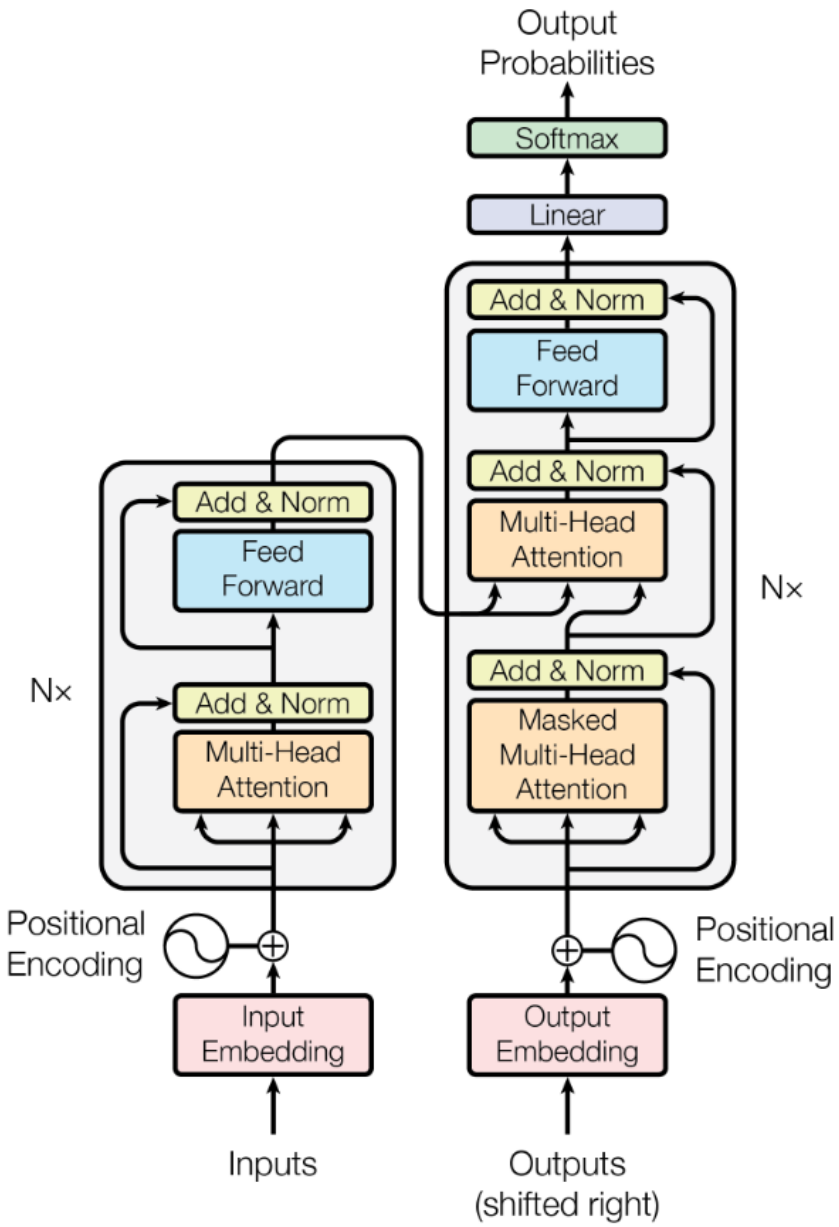

Figura 17: Arquitectura general del transformer. El codificador se encuentra a la izquierda y el decodificador a la derecha. [8]

A.4.1. Codificador

El codificador procesa las secuencias de entrada token por token. Cada token se incrusta inicialmente en un vector de dimensión d utilizando técnicas como Word2Vec. Después de la incrustación de palabras, se añade una codificación posicional que indica la posición absoluta y

relativa de los tokens en la secuencia. Esta codificación es crucial, ya que las partes posteriores del codificador son invariantes a la permutación [8].

El bloque de codificación incluye un mecanismo de autoatención que permite al modelo evaluar la relación entre diferentes tokens de la entrada. Finalmente, una red neuronal feed-forward sigue al bloque de atención y produce los tokens codificados uno por uno. Este bloque de codificación se puede repetir varias veces.

A.4.2. Decodificador

El decodificador funciona de manera similar al codificador pero opera en la secuencia de salida. Toma tanto los tokens codificados de entrada como los tokens de salida producidos hasta el momento para generar nuevos tokens. Incluye un bloque de autoatención y un bloque de atención cruzada que compara los tokens codificados de entrada con los tokens de salida. El bloque de decodificación también puede repetirse varias veces. Al final, una capa lineal y una activación softmax producen las probabilidades para todos los posibles tokens de salida [8].

A.4.3. Mecanismo de Atención

El mecanismo de atención permite al modelo asignar diferentes pesos de atención a diferentes tokens en una secuencia. Este mecanismo distingue tres entradas: la consulta (Query Q), la clave (Key K) y el valor (Value V). Cada token en una secuencia actúa como una entrada separada para Q, K o V. La atención se calcula como el producto punto escalado entre Q y K, seguido de una función softmax que normaliza los pesos de atención. Estos pesos se utilizan para seleccionar los valores correspondientes V [8].

En la **atención cruzada**, los tokens del codificador actúan como claves y valores, mientras que los tokens del decodificador actúan como consultas. Este mecanismo permite que el decodificador se enfoque en partes específicas de la secuencia de entrada mientras genera la secuencia de salida [2].

A.4.4. Atención Multi-Cabezal

La atención multi-cabezal repite el mecanismo de atención varias veces en paralelo. Antes de pasar los tokens de entrada a estos bloques de atención, se proyectan en incrustaciones más pequeñas. Cada cabeza de atención puede enfocarse en diferentes aspectos de la secuencia, como el sujeto y el objeto de una oración. Los resultados de las cabezas de atención se concatenan y se pasan a través de otra capa lineal [2].

A.5. Características de la máquina

A continuación se detallan las características de la máquina utilizada para llevar a cabo los experimentos y el procesamiento de datos en esta tesis:

1. **Procesador**: 12th Gen Intel(R) Core(TM) i7-12700F a 2.10 GHz
2. **Memoria RAM**: 32,0 GB (31,8 GB utilizable)
3. **Tipo de sistema**: Sistema operativo de 64 bits, procesador basado en x64
4. **Tarjeta gráfica**: NVIDIA GTX 1060 con 6 GB de memoria dedicada

A.6. Arquitectura de Modelos Conversores de Voz [12]

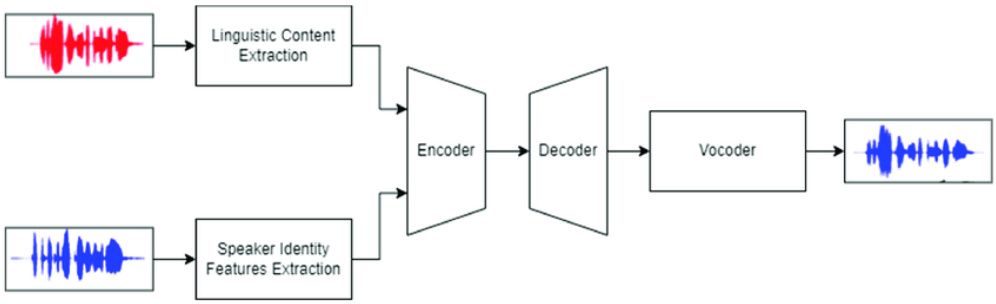

Figura 18: *Pipeline* típico de conversión de voz

A.6.1. Componentes de la Conversión de Voz

El proceso de conversión de voz se compone de varias etapas, cada una con un rol específico:

1. **Extracción de la Identidad del Hablante**: Este componente se encarga de extraer las características vocales del hablante. Utiliza técnicas como los vectores D, que capturan las características distintivas del hablante a partir de capas ocultas en redes neuronales profundas.
2. **Extracción del Contenido Lingüístico**: Procesa los datos del habla para obtener información temporal, como el contenido del discurso, el ritmo y la entonación. Este componente puede emplear modelos de reconocimiento automático del habla (ASR) y análisis de frecuencia fundamental (F0).
3. **Codificador (Encoder)**: Integra y representa las características extraídas de la identidad del hablante y el contenido lingüístico. Dado que las representaciones latentes del contenido lingüístico son dependientes del tiempo, el codificador se ocupa de combinar estas tareas.
4. **Decodificador/Vocoder**: Este componente convierte los datos procesados por el codificador en una banda sonora adecuada. A menudo, el input es un espectrograma, aunque en algunos casos el codificador y el vocoder se combinan para reducir las representaciones intermedias.

A.6.2. Técnicas Avanzadas en la Conversión de Voz

Los avances recientes han introducido varias técnicas innovadoras en la conversión de voz:

- **Modelos Generativos Antagonistas (GAN)**: Mejoran la calidad del audio convertido y permiten la conversión de voz entre múltiples hablantes sin necesidad de grandes conjuntos de datos.
- **Autoencoders Variacionales (VAE)**: Separan la identidad del hablante del contenido lingüístico, permitiendo que la voz del hablante objetivo suene como la del hablante fuente.
- **Redes Neuronales Convolucionales (CNN)**: Utilizadas en modelos como StarGAN v2, permiten la transferencia de características de estilo y emoción, además de la identidad del hablante.
- **Modelos de Cero-Shot**: Capaces de ajustar las características del hablante en una grabación para que coincidan con un hablante objetivo no visto previamente, sin necesidad de datos de entrenamiento paralelos.