Universidad de Valladolid

Escuela de Ingeniería Informática

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática
Mención en Computación

Voice Cloner: un bot en Telegram para la clonación de voz

Autor: Javier Cocho Cuesta

Universidad de Valladolid

Escuela de Ingeniería Informática

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática
Mención en Computación

Voice Cloner: un bot en Telegram para la clonación de voz

Autor: Javier Cocho Cuesta

Tutor: Valentín Cardeñoso Payo

Lo único que podemos decidir es qué hacer con el tiempo que se nos ha dado.

Agradecimientos

A los profesores, en especial a mi tutor Valentín, por todos los conocimientos transmitidos.

A mis padres y hermano, por aguantarme cada día.

A mis amigos y compañeros de clase, por hacer de mi paso por la facultad una experiencia inolvidable.

Y a mi novia, Alicia, por su apoyo incondicional y por ser la razón de intentar ser mejor cada día.

Resumen

En este trabajo se realiza un bot en Telegram que permite clonar la voz de una persona. Para lograr el objetivo se utilizarán técnicas de inteligencia artifial que permitan generar audios con réplicas de voces humanas, haciendo que generen voz a partir del texto indicado por el usuario. Para ello, se usan modelos y tecnologías de Deep Learning adecuados para la generación de voz a partir de texto usando modelos de codificación / descodificación, como el que propone Tortoise-TTS, que traslada técnicas generativas que se sabe han funcionado en el dominio de generación de imágenes a la generación de voz.

En este documento se recogen el análisis y diseño del bot, así como los aspectos teóricos necesarios para entender el funcionamiento del mismo.

Abstract

In this work, a bot is created in Telegram that allows cloning a person's voice. To achieve the objective, artificial intelligence techniques will be used that allow generating audio with replicas of human voices, causing them to generate voice from the text indicated by the user. To do this, Deep Learning models and technologies suitable for speech generation from text are used using coding/decoding models, such as the one proposed by Tortoise-TTS, which transfers generative techniques that are known to have worked in the generation domain. from images to speech generation.

This document contains the analysis and design of the bot, as well as the theoretical aspects necessary to understand how it works.

Índice general

Índice de cuadros

Índice de figuras

Capítulo 1

Introducción

1.1 Introducción

La clonación de voz consiste en la réplica de la voz humana de forma artifical. Actualmente, combina la inteligencia artificial con la tecnología de procesamiento de señales de audio, permitiendo recrear de manera realista las características propias de la voz de una persona. Este avance tecnológico ha despertado un creciente interés debido a sus diversas aplicaciones, que van desde la creación de voces personalizadas hasta la producción de contenido multimedia. A medida que la inteligencia artificial avanza, la capacidad para clonar la voz humana ha ido aumentando considerablemente, planteando importantes interrogantes éticos y de privacidad que necesitan ser abordados durante el desarrollo y la implementación de estas tecnologías.

El proceso de clonación de voz implica la utilización de modelos generativos, como las redes neuronales, para aprender y replicar los patrones acústicos y prosódicos propios de una persona. Este enfoque no solo busca imitar la entonación y el timbre de la voz, sino también capturar matices emocionales y expresivos que hacen que la voz humana sea única. A medida que se perfeccionan estas técnicas, surge la necesidad de explorar, además de los aspectos técnicos y algorítmicos, los éticos y sociales que acompañan a la clonación de voz.

En este contexto, el presente trabajo se sumerge en el mundo de la clonación de voz, explorando sus fundamentos teóricos, aplicaciones prácticas y desafíos éticos. Se creará un bot en Telegram con el uso de una tecnología que impulsa la clonación de voz, para así demostrar su potencial.

1.2 Motivación

La principal motivación detrás de este proyecto se debe a mi fascinación por todo lo relacionado con el sonido, así como su generación a través de inteligencia artificial, pudiendo ser manipulado mediante algoritmos para crear sonidos, tanto nuevos como recreaciones de otros.

El bot de este trabajo se presenta como un ejemplo de cómo la inteligencia artificial puede ser aplicada para replicar la voz humana. No solo se busca implementar esta tecnología, sino también conocer cómo se producen dichas réplicas, analizando los mecanismos que se utilizan para alcanzarlo.

1.3 Objetivos

El objetivo principal de este Trabajo de Fin de Grado es el de desarrollar un bot en Telegram que permita al usuario clonar la voz a partir de una muestra de audio, ya sea con una de las voces predefinidas del sistema o una guardada por el usuario.

Los distintos objetivos que se quieren conseguir son los siguientes:

- Clonar la voz humana a partir de una muestra de audio.
- Analizar y entender cómo funcionan los bots en Telegram.
- Controlar un bot de Telegram desde el lenguaje de programación Python.
- Mantener el bot constantemente activo.
- Realizar el bot de manera que sea sencillo de utilizar.
- Entender la importancia de respetar la privacidad de las personas (en este caso de su voz).

Así, surgen diversas tareas a realizar a lo largo del proyecto:

- Definir el trabajo y elaborar una planificación que permita llevarlo a cabo.
- Estudiar el problema desde un punto de vista teórico y después desarrollar un sistema.
- Realizar pruebas que permitan comprobar el correcto funcionamiento del sistema desarrollado.

1.4 Alcance

El objetivo principal de este trabajo puede resultar un poco ambicioso; al final la clonación de voz no es algo de lo que haya muchos desarrollos ni librerías implementadas para el lenguaje Python, por lo que la consecución de este dependerá de las tecnologías encontradas.

El resto de aspectos a tener en cuenta (la creación propiamente dicha del bot, que esté siempre activo...) no deberían suponer un gran problema a la hora de la realización de este trabajo.

1.5 Estructura de la memoria

Este Trabajo de Fin de Grado está dividido en los siguientes capítulos:

- **Capítulo 1 - Introducción**: donde se explica el contenido general del trabajo, así como los objetivos más importantes.
- **Capítulo 2 - Metodología**: donde se explica la metodología utilizada para la realización de esta proyecto, así como la planificación del mismo.
- **Capítulo 3 - Marco Conceptual**: donde se muestran los aspecto teóricos utilizados durante el desarrollo del bot.
- **Capítulo 4 - Análisis**: donde se detallan los requisitos y los casos de uso del bot.

- **Capítulo 5 - Diseño**: donde se muestran los diseños de los distintos diagramas.
- **Capítulo 6 - Implementación**: donde se explica la solución software propuesta.
- **Capítulo 7 - Pruebas**: donde se recogen diversos test realizados al bot.
- **Capítulo 8 - Conclusiones**: donde aparecen conclusiones obtenidas a lo largo del trabajo y se comentan líneas futuras de trabaajo.

Capítulo 2

Metodología

2.1 Metodología

Para el desarrollo de este proyecto se ha optado por utilizar una metodología ágil, ya que su uso está extendido en los equipos de trabajo dentro de la industria informática. Estas metodologías tienen una alta flexibilidad, lo que las hace más adecuadas para este tipo de proyectos que otras tradicionales.

Algunas de las metodologías ágiles más utilizadas son DSDM [30], Programación Extrema (XP) [31], Lean [29], y SCRUM [22]; para este proyecto se ha elegido esta última.

2.1.1 SCRUM

La metodología SCRUM se basa en la flexibilidad y en un desarrollo incremental del producto. En él, el desarrollo del producto está basado en iteraciones conocidas como sprints de duración fija, habitualmente de dos a cuatro semanas, en los que un equipo se enfoca en una parte concreta del producto. Después de cada sprint, se debe haber creado un incremento del producto que sea funcional, es decir, que funcione y cumpla con los objetivos establecidos para ese sprint.

En la metodología SCRUM, hay tres roles diferenciados:

- Product Owner: responsable de la creación de una lista de requisitos o características que se deben desarrollar, ya que es quien realmente conoce el negocio y entiende cuál es el producto final que solicita el cliente.

- Scrum Master: persona que actúa como facilitador del proceso SCRUM, y encargado de que se cumplan las reglas establecidas, asegurándose de que el equipo también las entienda.

- Equipo de desarrollo: encargado del desarrollo del producto, produciendo sus diversos incrementos.

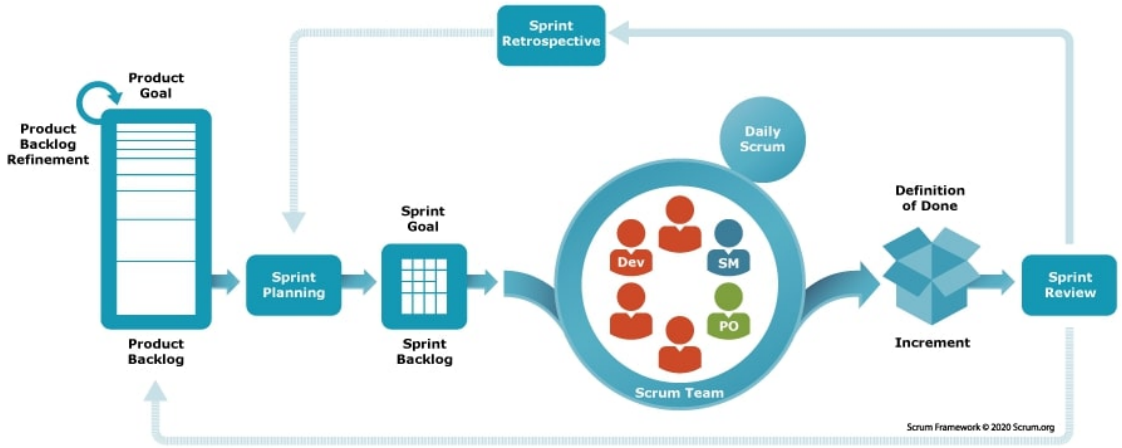


Figura 2.1: SCRUM [22]

2.2 Análisis de riesgos

En este apartado se muestran los riesgos posibles que se pueden encontrar durante la realización de este proyecto. Un riesgo es un evento que en caso de ocurrir, afecta a al menos un objetivo del proyecto [12], ya sea en el costo o tiempo del mismo.

El objetivo del análisis de riesgos es el de prever esos riesgos para tratarlos y reducir el posible impacto que tengan sobre el proyecto. Para analizarlos correctamente, se mide de cada uno una probabilidad de ocurrencia y un nivel de impacto, asignando un nivel según los intervalos que se muestran a continuación [21]:

- Probabilidad:

Nivel de probabilidad	Rango
Alto	Más de un 50 % de probabilidad de ocurrencia
Significante	30 % - 50 % de probabilidad de ocurrencia
Moderado	10 % - 29 % de probabilidad de ocurrencia
Bajo	Menos de un 10 % de probabilidad de ocurrencia

- Impacto:

Nivel de impacto	Rango
Alto	Más de un 30 % por encima del presupuesto
Significante	20 % - 29 % por encima del presupuesto
Moderado	10 % - 19 % por encima del presupuesto
Bajo	Menos de un 10 % por encima del presupuesto

A continuación se muestran los principales riesgos detectados junto con sus respectivos planes de prevención y contingencia:

Identificador	R01 - Retraso en la planificación de las tareas
Descripción	La duración estimada de las tareas no se ajusta a la realidad
Nivel de probabilidad	Moderado
Nivel de impacto	Alto
Plan de prevención	Realizar una planificación más holgada
Plan de contingencia	Replanificar las tareas, priorizando las más importantes y, si es necesario, incrementar el número de horas invertidas en el proyecto

Cuadro 2.1: R01 - Retraso en la planificación de las tareas

Identificador	R02 - Indisponibilidad del alumno
Descripción	El alumno puede no estar disponible por motivos de salud o de otro tipo que pueden afectar al desarrollo del proyecto
Nivel de probabilidad	Alto
Nivel de impacto	Alto
Plan de prevención	Mantener un estilo de vida saludable durante el desarrollo del proyecto
Plan de contingencia	Replanificar las tareas

Cuadro 2.2: R02 - Indisponibilidad del alumno

Identificador	R03 - Indisponibilidad del tutor
Descripción	El tutor no se encuentra disponible debido a motivos personales, de salud, o de otros proyectos desarrollados en paralelo
Nivel de probabilidad	Significante
Nivel de impacto	Significante
Plan de prevención	Acordar reuniones con antelación
Plan de contingencia	Replanificar tareas para avanzar en aquellas que no dependan del tutor

Cuadro 2.3: R03 - Indisponibilidad del tutor

Identificador	R04 - Problemas con la máquina virtual que aloja el código
Descripción	La máquina virtual no funciona o se retrasa el montaje de ella
Nivel de probabilidad	Bajo
Nivel de impacto	Medio
Plan de prevención	Almacenar todos los archivos en local para que, en caso de pérdida de ficheros, no retrase aún más el proyecto
Plan de contingencia	Replanificar tareas para realizar aquellas que no dependan de la máquina virtual y, en caso necesario, consultar con los técnicos de la escuela

Cuadro 2.4: R04 - Problemas con la máquina virtual que aloje el código

Identificador	R05 - Desconocimiento de las tecnologías utilizadas
Descripción	El alumno no conoce las tecnologías usadas y requiere de más tiempo para saber utilizarlas
Nivel de probabilidad	Alto
Nivel de impacto	Moderado
Plan de prevención	Investigar previamente sobre dichas nuevas tecnologías
Plan de contingencia	Replanificar las tareas que dependan de tecnologías nuevas de forma más holgada

Cuadro 2.5: R05 - Desconocimiento de las tecnologías utilizadas

Identificador	R06 - Planteamiento y/o implementación erróneos de las soluciones
Descripción	Las soluciones propuestas no son correctas
Nivel de probabilidad	Bajo
Nivel de impacto	Alto
Plan de prevención	Desarrollo incremental
Plan de contingencia	Volver a implementar las partes mal implementadas, replanificando si es necesario

Cuadro 2.6: R06 - Errores en la implementación del bot

Identificador	R07 - Problemas con el equipo utilizado
Descripción	El equipo utilizado para el desarrollo del proyecto falla
Nivel de probabilidad	Bajo
Nivel de impacto	Alto
Plan de prevención	Almacenar todos los ficheros en una nube, para no depender del equipo en el que se desarrolla el proyecto
Plan de contingencia	Solicitar un equipo a algún amigo o familiar para poder continuar con el proyecto

Cuadro 2.7: R07 - Problemas con el equipo utilizado

Identificador	R08 - Problemas con la compaginación con el trabajo
Descripción	El alumno no dispone del tiempo suficiente para realizar el trabajo según el tiempo previsto
Nivel de probabilidad	Alto
Nivel de impacto	Alto
Plan de prevención	Planificación realista de las tareas
Plan de contingencia	Replanificar las tareas

Cuadro 2.8: R08 - Problemas con la compaginación con el trabajo

2.3 Recursos

Los recursos necesarios para la realización de este proyecto son los siguientes:

- Tecnológicos: el equipo utilizado por el desarrollador, el software, la conexión a internet, luz y la máquina virtual *tauro* de la Escuela de Ingeniería Informática de la UVa.
- Humanos: el alumno, que actuará de desarrollador en la totalidad del proyecto.

2.4 Hardware y software

A continuación se muestra el hardware y software utilizados para la realización del proyecto:

Nombre	**Descripción**
Visual Studio Code	Editor de código fuente
Overleaf	Editor online para LaTeX
Máquina virtual de la Escuela	Máquina *tauro*
Ordenador portátil	Ordenador de gama media
Ordenador de sobremesa	Ordenador de gama alta

Cuadro 2.9: Fases de desarrollo del proyecto previstas.

2.5 Planificación inicial

Las tareas se agruparon en los siguientes bloques a modo de planificación inicial:

- Análisis del problema: estudio e investigación acerca del problema, con la posibilidad del análisis de soluciones ya existentes.
- Diseño del bot: realización del diseño del bot mediante los diagramas necesarios.
- Implementación del bot: desarrollo del código que compone el bot.
- Revisión del bot: realización de pruebas para comprobar el correcto funcionamiento del bot.
- Elaboración de la memoria: redacción de la memoria de trabajo.

Posteriormente, se realizó un diagrama de descomposición de tareas WBS, que es un enfoque que consiste en detectar las tareas principales y dividirlas a su vez en subtareas.

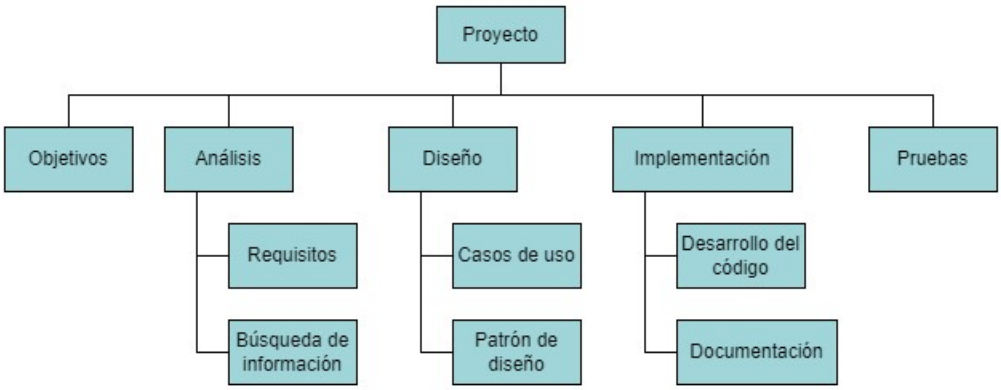


Figura 2.2: Diagrama de tareas

2.6 Fases

El desarrollo del proyecto se dividió en sprints, de la siguiente forma:

Sprint	Nombre de actividad	Semanas
Sprint 1	Preparación del entorno de programación	1-2
Sprint 1	Estudio del funcionamiento de bots	1-2
Sprint 1	Estudio de conexión de Telegram con Python	1-2
Sprint 2	Elaboración inicial del código	3-4
Sprint 2	Estudio de la clonación	3-4
Sprint 3	Búsqueda de soluciones similares	5-6
Sprint 3	Desarrollo de la conexión Telegram-Python	5-6
Sprint 4	Comienzo de la elaboración de requisitos y casos de uso	7-8
Sprint 4	Recogida del audio en código	7-8
Sprint 5	Investigación clonación en código	9-10
Sprint 5	Conexión de TorToise con bot en Python	9-10
Sprint 6	Finalización del código	11-12
Sprint 7	Elaboración de la memoria	13-14
Sprint 7	Puesta en marcha de la máquina virtual	13-14
Sprint 8	Finalización de la memoria	15

Cuadro 2.10: Sprints previstos

A continuación, se muestra el diagrama de Gantt de actividades, realizado en Free Online Gantt Chart Software [11]:

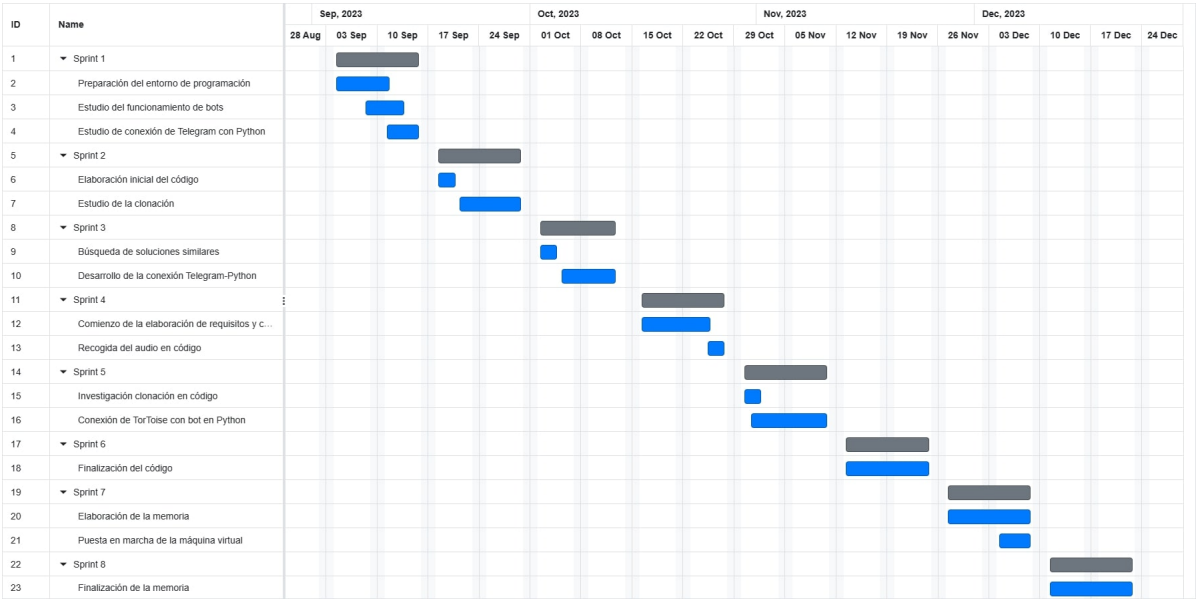


Figura 2.3: Diagrama de Gantt de actividades

La realización de los dos últimos sprint (7 y 8) se alargó más allá de la planificación inicial, acabando estos en el mes de mayo, debido a la indisponibilidad del alumno durante diciembre y los primeros meses de 2024.

2.7 Costes

A continuación se muestra un análisis de los costes estimados para la realización del proyecto. Para ello se tendrá en cuenta que en el proyecto se han invertido 75 días, con una media de trabajo de 4 horas por día.

Primero, en cuanto al hardware, dado un tiempo medio de vida útil de hardware de dos años, tenemos la siguiente tabla de costes de cada componente:

Componente	**Coste original**	**Coste en el proyecto**
Versus PC Expert Qubist Silver Intel®Core i7™	1,163.34 €	119.52 €
HP Pavilion Intel®Core i5™	650 €	66.78 €
Monitor PC LG 24MK600M	99 €	10,17 €
Coste hardware	1,912.34 €	**196.47 €**

Cuadro 2.11: Coste estimado del hardware

En cuanto al coste de personal, se ha tenido en cuenta el salario medio de un ingeniero informático en España, que se estima en 2,250 € al mes, que equivale a 13.85 € la hora.

El coste de software en este proyecto es de 0 €.

Además, hay que sumar otros costes indirectos como el de internet y luz que en tres meses, con las tarifas contratadas, se estima en 90 €.

Por tanto, el coste total estimado del proyecto es el siguiente:

Concepto	**Coste estimado**
Hardware	196.47 €
Personal	4,155 €
Indirectos	90 €
Total	**4,441.47 €**

Cuadro 2.12: Coste estimado total

Capítulo 3

Marco Conceptual

En este capítulo se muestra el marco teórico en el que se sustenta este trabajo, para así explicar los distintos conceptos sobre los que se ha trabajado.

3.1 Telegram

Telegram es una app de mensajería instantánea creada en Rusia que permite a sus usuarios enviar mensajes de texto, imágenes, vídeos y más a través de Internet.

Figura 3.1: Logo de Telegram [32]

Una de sus características más destacadas es su énfasis en la privacidad y en la seguridad; esto lo consigue utilizando cifrado de extremo a extremo en las conversaciones, lo que hace que solo las personas pertenecientes a esa conversación puedan acceder a los mensajes.

Además de los chats privados, Telegram también permite la creación de grupos y de canales, para así permitir la comunicación directa entre varias personas a la vez.

Adicionalmente, cuenta con una API abierta que ha facilitado e impulsado mejoras en ella, así como el desarrollo de numerosos bots.

3.2 Bot conversacional

Un bot conversacional (*chatbot* en inglés) es una aplicación software que, mediante respuestas automáticas, simula tener una conversación con una persona humana, utilizando el razonamiento basado en casos.

Si bien históricamente la conversación se realizaba mediante texto, últimamente están surgiendo chatbots que permiten la comunicación mediante audio.

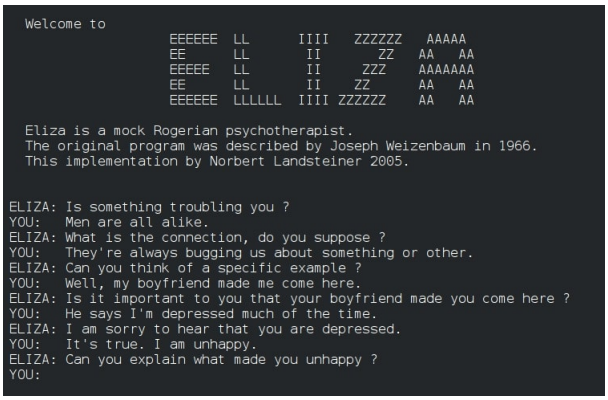


Figura 3.2: Chatbot ELIZA, uno de los primeros chatbots (1966) [26]

Los chatbots tienen un claro objetivo: mejorar la experiencia de uso de un usuario dentro de un servicio ya sea, por ejemplo, una página web o una aplicación, en este caso Telegram.

3.3 Bots en Telegram

Los bots en Telegram están diseñados para interactuar con los usuarios de la aplicación de forma automática a través de un conjunto de comandos predefinidos o mediante respuestas automáticas a palabras clave. Para crear esos bots, se puede utilizar la *Bot API* propia de Telegram.

Los bots son muy sencillos de utilizar; basta con escribir el nombre del bot en el buscador de Telegram y la aplicación te lo muestra. Esto, junto con que no necesita instalación alguna y que Telegram es una aplicación multiplataforma, facilita mucho el acceso y el uso de los bots.

3.4 Procesamiento del lenguaje natural

El procesamiento del lenguaje natural (NLP) es una rama de la inteligencia artificial que permite a los ordenadores entender el lenguaje humano, ya sea oral o escrito, y también imitarlo.

El NLP integra el análisis lingüístico y su comprensión, junto con modelos informáticos y estadísticos de aprendizaje automático y aprendizaje profundo, lo que posibilita tanto el entendimiento como la generación.

El NLP tiene dos ramas diferenciadas: la comprensión del lenguaje natural (NLU), y la generación de lenguaje natural (NLG).

3.5 Clonación de voz

La clonación de voz consiste en el proceso de replicar o imitar la voz de una persona mediante el uso de tecnologías avanzadas. Debido a los avances en inteligencia artificial y procesamiento de audio, este fenómeno ha ido ganando atención con el paso de los años.

El proceso consiste normalmente en el entrenamiento de un modelo de inteligencia artificial con muestras de voz de la persona objetivo; una vez entrenado dicho modelo, se pueden obtener nuevas grabaciones de voz que suenen muy parecidas a la voz original.

Para el proceso de clonación de voz, utilizaremos una solución existente conocida como TorToiSe-TTS [3, 9].

3.6 Arquitectura de transformadores

Antes de entrar a ver TorToiSe, conviene entender qué son y cómo funcionan los transformadores. Los transformadores son la arquitectura de redes neuronales más utilizada actualmente en NLP, e introducen el concepto de la *atención* (de ahí que el paper en el que se publicaron se llamara *Attention Is All You Need* [2]).

Los mecanismos de atención permiten dar contexto a las palabras dentro de una secuencia mediante una valoración numérica de las conexiones entre dichas palabras.

La arquitectura de transformadores, mostrada en la figura 3.3, está basada en una estructura codificador-decodificador, que funciona de la siguiente manera:

1. El codificador convierte la entrada, que es una secuencia de símbolos, en una secuencia de valores continuos.

2. El decodificador, a partir de los valores generados por el codificador, genera una secuencia de salida de símbolos.

3. La salida del decodificador sirve de entrada de nuevo al codificador, repitiéndose el proceso de forma iterativa.

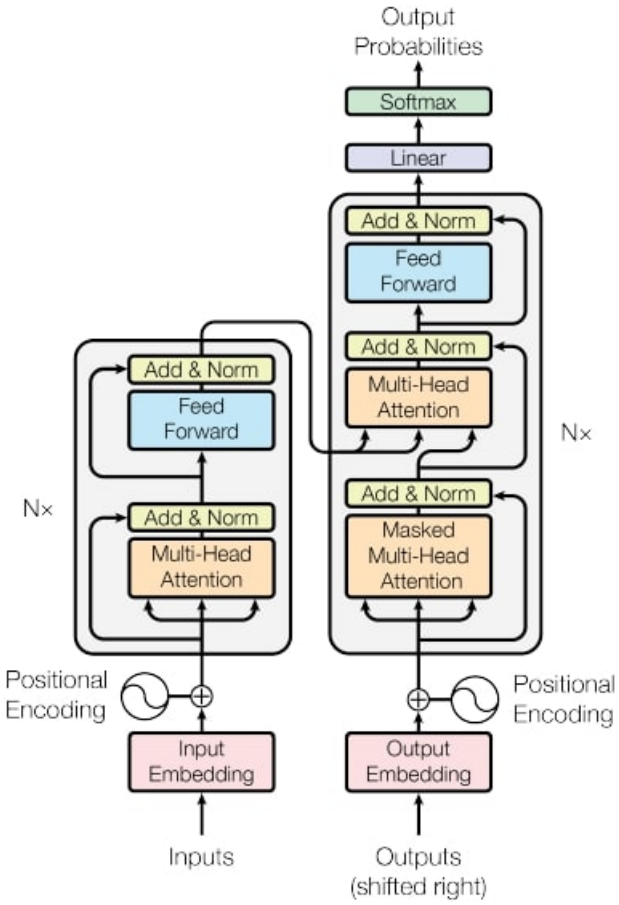


Figura 3.3: Arquitectura de transformadores[5]

3.7 TorToiSe

TorToiSe [3, 8] es un programa de conversión de texto a voz (*text-to-speech*, TTS) creado por James Betker, capaz de imitar voces humanas a partir de apenas tres pequeñas muestras de dichas voces. Está formado por cinco redes neuronales entrenadas de forma independiente que se combinan para obtener como resultado una voz clonada a partir de las voces introducidas y el texto que se quiere reproducir, tal y como se ilustra en la figura 3.4.

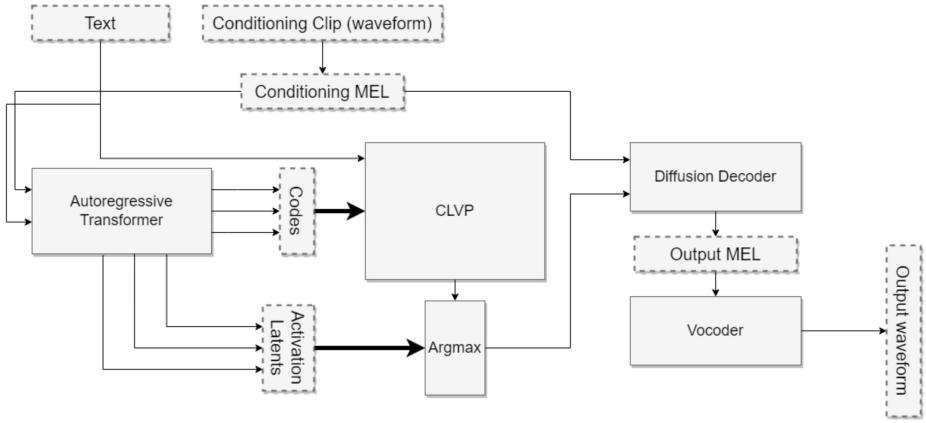


Figura 3.4: Arquitectura de TorToiSe, tomada de [3]

A continuación se explican las distintas redes que conforman TorToiSe.

3.7.1 Autoregressive Decoder

Al modelo se le proporcionan unas muestras de audio de referencia para conseguir imitar correctamente la voz, teniendo en cuenta la entonación y la prosodia (estudio de los rasgos fónicos que afectan a la métrica, especialmente de los acentos y de la cantidad). Estas muestras son procesados por una red convolucional reductora, seguida por una pila de transformadores de atención plena.

La arquitectura final es de un solo decodificador, un modelo GPT-2 [6] como el usado por DALL·E, modificado ligeramente.

El autoregressive decoder es el principal encargado de la traducción de texto a voz.

3.7.2 CLVP

Para obtener buenos resultados en los modelos generativos, se puede utilizar un discriminador cualitativo que se utilice para clasificar los resultados obtenidos de él y quedarse con aquellos que sean mejores.

El modelo *Contrastive Language-Voice Pretraining* (CLVP) se encarga de detectar las salidas del autoregressive decoder en las que la pérdida de información debido a la compresión realizada sea notable, para posteriormente quedarse solo con aquellas que tengan buena calidad. Consume las salidas

tokenizadas del autoregressive decoder para decodificarlas mediante una pila de doce capas de transformadores de atención plena.

Su principal función es la de conducir al autoregressive decoder hacia salidas en las que el texto que se ha introducido esté bien reproducido por la voz.

3.7.3 CVVP

Contrastive voice-voice pretraining (CVVP) es un modelo de contraste que ha aprendido a emparejar una muestra de audio de una persona hablando algo con las salidas tokenizadas del autoregressive decoder. Su función principal es la de conducir al autoregressive decoder hacia salidas que tengan una voz similar a la de la muestra de referencia.

3.7.4 Diffusion Decoder

El diffusion decoder es un modelo de difusión (actualmente, los más usados en la realización de modelos generativos) que toma las salidas del autoregressive decoder y las muestras de referencia y las utiliza para construir un espectrograma MEL real, ya que en la salida del autoregressive decoder las representaciones del habla están altamente comprimidas y hay que decodificarlas en forma de ondas reales que puedan reproducirse en un altavoz.

Las salidas autorregresivas se preprocesan primero mediante una pila de cuatro bloques de atención plena y tres bloques adicionales de combinación de atención plena/red con infusión de pasos temporales.

Las muestras de referencia se preprocesan en un punto latente con una pila de transformadores estilo ViT; la salida de este módulo se denomina latente de acondicionamiento.

La latente de acondicionamiento se utiliza para escalar y desplazar la media y la varianza de los códigos latentes. A continuación, estos códigos mutados se concatenan en la entrada del diffusion decoder antes de pasar por la pila de transformadores principal descrita anteriormente.

3.7.5 Vocoder

Con todo lo anterior, tenemos un modelo que puede producir espectrogramas MEL cuando se le da un texto y algunos datos de voz de referencia. El último paso es convertir ese espectograma MEL en audio.

Para este sistema, se ha usado un vocoder de código abierto llamado *uvivnet* [33].

3.7.6 Funcionamiento general

Una vez vistos todos los componentes que forman el sistema de forma individual, podemos explicar su funcionamiento a nivel global:

1. El autoregressive decoder toma entradas de texto y clips de referencia, y emite latentes y los correspondientes códigos de token que representan datos de audio altamente comprimidos.
2. Los modelos CLVP y CVVP eligen el mejor candidato:
   a) CLVP produce una puntuación de similitud entre el texto de entrada y cada secuencia de códigos candidata.
   b) CVVP produce una puntuación de similitud entre los clips de referencia y cada candidato.
3. El diffusion decoder consume las latentes autorregresivas y los clips de referencia para producir un espectrograma MEL que representa alguna salida de voz.
4. El vocoder univnet transforma el espectrograma MEL en audio real.

Capítulo 4

Análisis

En este capítulo se detallan las distintas partes del análisis del proyecto.

4.1 Requisitos

En este apartado se indican los distintos requisitos del bot, divididos en tres tipos:

- Requisitos funcionales: aquellas funciones que deben estar implementadas en el bot.
- Requisitos no funcionales: relacionados con las características y restricciones del bot.
- Requisitos de información: los distintos datos con los que se trabaja.

A continuación se muestran las tablas con los distintos requisitos.

4.1.1 Requisitos funcionales

Código	Requisito	Descripción
RF-01	start	El bot devolverá un mensaje de bienvenida al usuario en el que le mostrará las funcionalidades del bot.
RF-02	help	El bot mostrará una lista de los comandos y su descripción.
RF-03	cloningmode	El bot recibe un mensaje de voz o texto y devuelve mensaje de voz con lo dicho con la voz de la persona seleccionada.
RF-04	uploadingmode	El bot se prepara para recibir un audio
RF-05	ultrafast	El modo de clonación se establece en ultrafast.
RF-06	fast	El modo de clonación se establece en fast.

RF-07	standard	El modo de clonación se establece en standard.
RF-08	highquality	El modo de clonación se establece en highquality.
RF-09	cristiano	La voz de clonación se establece en la de Cristiano Ronaldo.
RF-10	deniro	La voz de clonación se establece en la de Robert De Niro.
RF-11	customvoices	Se muestran las voces guardadas por el usuario para seleccionar una.
RF-12	deletevoice	Se muestran las voces guardadas por el usuario para borrar una.
RF-13	seleccionar custom	Al seleccionar una custom, se selecciona esa voz para ser clonada.
RF-14	eliminar custom	Al seleccionar una custom, se elimina esa voz.
RF-15	guardar custom	Se guarda una voz con el nombre introducido por el usuario.

Cuadro 4.1: Requisitos funcionales

4.1.2 Requisitos no funcionales

Código	Requisito	Descripción
RNF-01	Multiusuario	El bot debe permitir ser usado por distintos usuarios de forma simultánea.
RNF-02	Multiplataforma	El bot debe permitir ser usado desde cualquier dispositivo desde el que se ejecute Telegram.
RNF-03	Clonación de voz	El bot recibe un mensaje de texto o voz y lo devuelve clonado.

Cuadro 4.2: Requisitos no funcionales

4.2 Modelo de dominio

A continuación se muestra una imagen sencilla del modelo de dominio. Su función es mostrar las diversas entidades que conforman el sistema, junto con los atributos que contienen y las relaciones entre ellas.

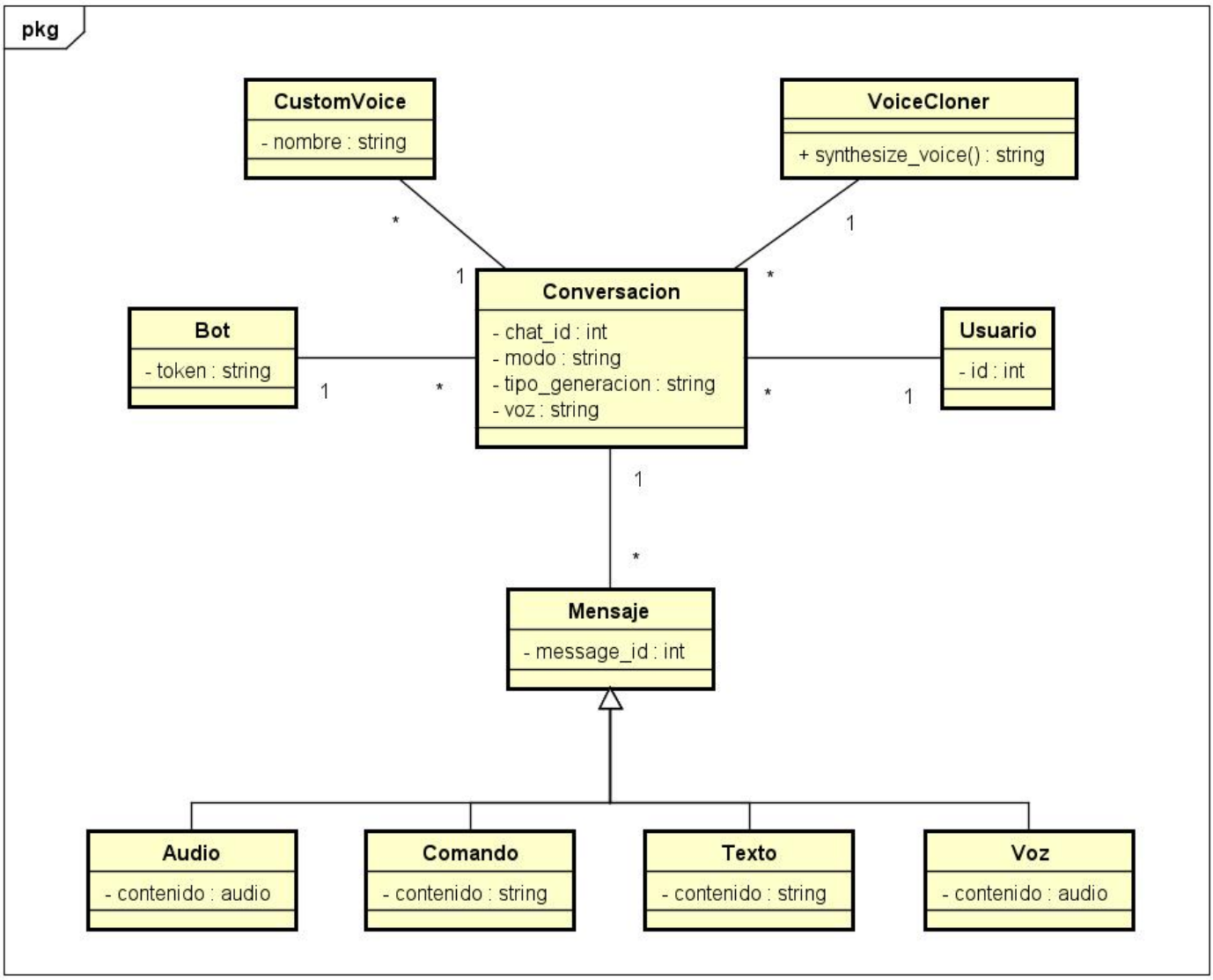


Figura 4.1: Modelo de dominio

4.3 Casos de uso

4.3.1 Diagrama de casos de uso

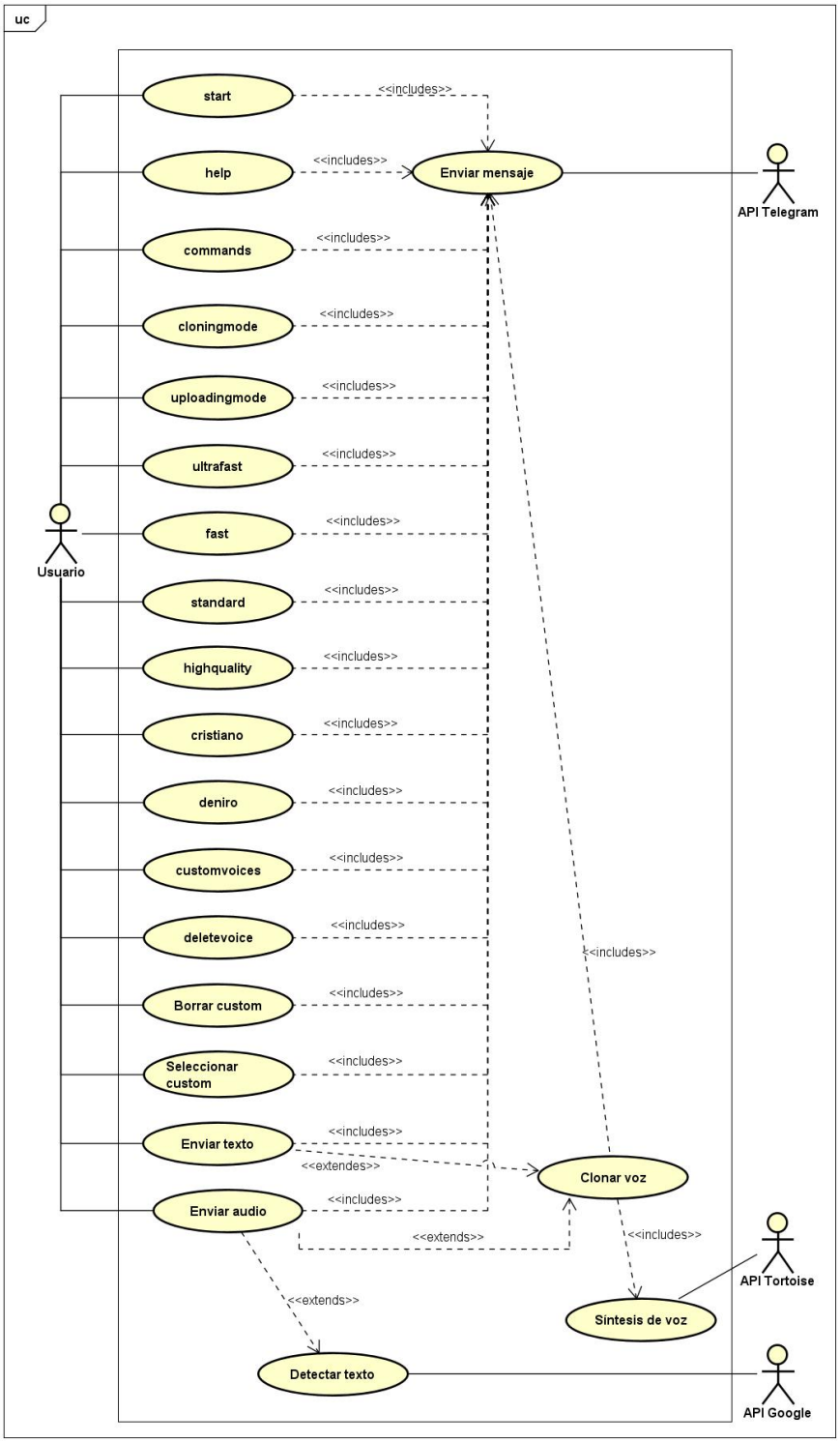


Figura 4.2: Diagrama de casos de uso

4.3.2 Descripción de casos de uso

Para un entendimiento mejor de los casos de uso, se muestran a continuación varias variables globales que se han utilizado en el desarrollo del bot:

- *CLONING*: indica si el próximo audio que se envía es para clonar su contenido (True) o el objetivo es guardarlo como custom (False).
- *DELETING*: en True, indica si se está esperando un comando que indique la voz de la persona guardada que se quiere eliminar.
- *MODE*: indica el modo (velocidad/calidad) de clonación.
- *NAME*: indica el nombre de la persona de la que se quiere clonar la voz.
- *WAITING_FOR_CUSTOM*: en True, indica si se está esperando un comando que indique la voz de la persona que se quiere clonar.

Los casos de uso son los siguientes:

Identificador	**CU-01: start**
Actor principal	Usuario
Actor implicado	API Telegram
Descripción	El usuario escribe /start y recibe un mensaje de bienvenida
Precondiciones	1. El usuario ha iniciado un chat con el bot
Secuencia principal	1. El usuario envía /start. 2. El sistema detecta el comando. 3. El sistema envía un mensaje de bienvenida al usuario
Postcondiciones	1. El sistema ha enviado un mensaje por el chat.

Cuadro 4.3: CU-01: start

Identificador	**CU-02: help**
Actor principal	Usuario
Actor implicado	API Telegram
Descripción	El usuario escribe /help y recibe un mensaje de ayuda
Precondiciones	1. El usuario ha iniciado un chat con el bot
Secuencia principal	1. El usuario envía /help. 2. El sistema detecta el comando. 3. El sistema envía un mensaje mostrando cómo se utiliza el bot.
Postcondiciones	1. El sistema ha enviado un mensaje por el chat.

Cuadro 4.4: CU-02: help

Identificador	**CU-03: commands**
Actor principal	Usuario
Actor implicado	API Telegram
Descripción	El usuario escribe /commands y recibe un mensaje con la lista de todos los comandos disponibles
Precondiciones	1. El usuario ha iniciado un chat con el bot
Secuencia principal	1. El usuario envía /commands. 2. El sistema detecta el comando. 3. El sistema envía un mensaje mostrando la lista de comandos disponibles.
Postcondiciones	1. El sistema ha enviado un mensaje por el chat.

Cuadro 4.5: CU-03: commands

Identificador	**CU-04: cloningmode**
Actor principal	Usuario
Actor implicado	API Telegram
Descripción	El usuario escribe /cloningmode y el sistema pasa a modo clonación
Precondiciones	1. El usuario ha iniciado un chat con el bot
Secuencia principal	1. El usuario envía /cloningmode. 2. El sistema detecta el comando. 3. El sistema pasa a modo de clonación, preparándose así para que el próximo mensaje recibido sea con el fin de ser el texto a replicar. 4. El sistema envía un mensaje indicando que se ha cambiado a modo clonación.
Postcondiciones	1. El sistema ha enviado un mensaje por el chat. 2. El valor de CLONING ha cambiado a True.

Cuadro 4.6: CU-04: cloningmode

Identificador	**CU-05: uploadingmode**
Actor principal	Usuario
Actor implicado	API Telegram
Descripción	El usuario escribe /uploadingmode y el sistema pasa a modo subida
Precondiciones	1. El usuario ha iniciado un chat con el bot
Secuencia principal	1. El usuario envía /uploadingmode. 2. El sistema detecta el comando. 3. El sistema pasa a modo de subida, preparándose así para recibir un mensaje de voz para guardar en el sistema. 4. El sistema envía un mensaje indicando que se ha cambiado a modo subida.
Postcondiciones	1. El sistema ha enviado un mensaje por el chat. 2. El valor de CLONING ha cambiado a False.

Cuadro 4.7: CU-05: uploadingmode

Identificador	**CU-06: ultrafast**
Actor principal	Usuario
Actor implicado	API Telegram
Descripción	El usuario escribe /ultrafast y el sistema cambia el modo de clonación a ultrafast
Precondiciones	1. El usuario ha iniciado un chat con el bot
Secuencia principal	1. El usuario envía /ultrafast. 2. El sistema detecta el comando. 3. El sistema cambia el modo de clonación a ultrafast. 4. El sistema envía un mensaje indicando que se ha cambiado a modo de clonación ultrafast.
Postcondiciones	1. El sistema ha enviado un mensaje por el chat. 2. El valor de MODE ha cambiado a 'ultrafast'.

Cuadro 4.8: CU-06: ultrafast

Identificador	**CU-07: fast**
Actor principal	Usuario
Actor implicado	API Telegram
Descripción	El usuario escribe /fast y el sistema cambia el modo de clonación a fast
Precondiciones	1. El usuario ha iniciado un chat con el bot
Secuencia principal	1. El usuario envía /fast. 2. El sistema detecta el comando. 3. El sistema cambia el modo de clonación a fast. 4. El sistema envía un mensaje indicando que se ha cambiado a modo de clonación fast.
Postcondiciones	1. El sistema ha enviado un mensaje por el chat. 2. El valor de MODE ha cambiado a 'fast'.

Cuadro 4.9: CU-07: fast

Identificador	**CU-08: standard**
Actor principal	Usuario
Actor implicado	API Telegram
Descripción	El usuario escribe /standard y el sistema cambia el modo de clonación a standard
Precondiciones	1. El usuario ha iniciado un chat con el bot
Secuencia principal	1. El usuario envía /standard. 2. El sistema detecta el comando. 3. El sistema cambia el modo de clonación a standard. 4. El sistema envía un mensaje indicando que se ha cambiado a modo de clonación standard.
Postcondiciones	1. El sistema ha enviado un mensaje por el chat. 2. El valor de MODE ha cambiado a 'standard'.

Cuadro 4.10: CU-08: standard

Identificador	**CU-09: highquality**
Actor principal	Usuario
Actor implicado	API Telegram
Descripción	El usuario escribe /highquality y el sistema cambia el modo de clonación a highquality
Precondiciones	1. El usuario ha iniciado un chat con el bot
Secuencia principal	1. El usuario envía /highquality. 2. El sistema detecta el comando. 3. El sistema cambia el modo de clonación a highquality. 4. El sistema envía un mensaje indicando que se ha cambiado a modo de clonación highquality.
Postcondiciones	1. El sistema ha enviado un mensaje por el chat. 2. El valor de MODE ha cambiado a 'highquality'.

Cuadro 4.11: CU-09: highquality

Identificador	**CU-10: cristiano**
Actor principal	Usuario
Actor implicado	API Telegram
Descripción	El usuario escribe /cristiano y el sistema cambia la voz de clonación a la de Cristiano Ronaldo
Precondiciones	1. El usuario ha iniciado un chat con el bot
Secuencia principal	1. El usuario envía /cristiano. 2. El sistema detecta el comando. 3. El sistema cambia la voz de clonación a la de Cristiano Ronaldo. 4. El sistema envía un mensaje indicando que se va a clonar la voz de Cristiano Ronaldo.
Postcondiciones	1. El sistema ha enviado un mensaje por el chat. 2. El valor de NAME ha cambiado a 'cristiano'.

Cuadro 4.12: CU-10: cristiano

Identificador	**CU-11: deniro**
Actor principal	Usuario
Actor implicado	API Telegram
Descripción	El usuario escribe /deniro y el sistema cambia la voz de clonación a la de Robert De Niro
Precondiciones	1. El usuario ha iniciado un chat con el bot
Secuencia principal	1. El usuario envía /deniro. 2. El sistema detecta el comando. 3. El sistema cambia la voz de clonación a la de Robert De Niro. 4. El sistema envía un mensaje indicando que se va a clonar la voz de Robert De Niro.
Postcondiciones	1. El sistema ha enviado un mensaje por el chat. 2. El valor de NAME ha cambiado a 'deniro'.

Cuadro 4.13: CU-11: deniro

Identificador	**CU-12: customvoices**
Actor principal	Usuario
Actor implicado	API Telegram
Descripción	El usuario escribe /customvoices y el sistema muestra una lista con las voces guardadas
Precondiciones	1. El usuario ha iniciado un chat con el bot
Secuencia principal	1. El usuario envía /customvoices. 2. El sistema detecta el comando. 3. El sistema busca las voces guardadas por el usuario. 4. El sistema envía un mensaje mostrando las voces que ha encontrado.
Postcondiciones	1. El sistema ha enviado un mensaje por el chat. 2. El valor de WAITING_FOR_CUSTOM ha cambiado a TRUE.

Cuadro 4.14: CU-12: customvoices

Identificador	**CU-13: deletevoice**
Actor principal	Usuario
Actor implicado	API Telegram
Descripción	El usuario escribe /deletevoice y el sistema muestra una lista con las voces guardadas
Precondiciones	1. El usuario ha iniciado un chat con el bot
Secuencia principal	1. El usuario envía /deletevoice. 2. El sistema detecta el comando. 3. El sistema busca las voces guardadas por el usuario. 4. El sistema envía un mensaje mostrando las voces que ha encontrado.
Postcondiciones	1. El sistema ha enviado un mensaje por el chat. 2. El valor de DELETING ha cambiado a TRUE.

Cuadro 4.15: CU-13: deletevoice

Identificador	**CU-14: Borrar custom**
Actor principal	Usuario
Actor implicado	API Telegram
Descripción	El usuario eliga una de entre las voces que ha guardado para clonarla
Precondiciones	1. El usuario ha iniciado un chat con el bot 2. El usuario ha guardado una voz anteriormente 3. El usuario ha escrito previamente /deletevoice
Secuencia principal	1. El usuario escribe '/' seguido del nombre de la voz guardada que quiere eliminar. 2. El sistema detecta el comando. 3. El sistema busca las voces guardadas por el usuario. 4. El sistema envía un mensaje indicando que se ha eliminado la voz de la persona elegida.
Secuencia alternativa	1.1.a. El usuario envía un mensaje de texto que no es ningún comando predeterminado ni ninguna custom. 1.1.b. El sistema envía un mensaje diciendo informando que el comando introducido es incorrecto y que vuelva a enviar /deletevoice si el usuario quiere eliminar una voz. 1.2.a. El usuario envía un mensaje de audio. 1.2.b. El sistema envía un mensaje indicando que no tiene que enviar ningún audio, sino enviar un comando.
Postcondiciones	1. El sistema ha enviado un mensaje por el chat. 2. El valor de DELETING ha cambiado a TRUE.

Cuadro 4.16: CU-14: Borrar custom

Identificador	**CU-15: Seleccionar custom**
Actor principal	Usuario
Actor implicado	API Telegram
Descripción	El usuario eliga una de entre las voces que ha guardado para clonarla
Precondiciones	1. El usuario ha iniciado un chat con el bot 2. El usuario ha guardado una voz anteriormente 3. El usuario ha escrito previamente /customvoices
Secuencia principal	1. El usuario escribe '/' seguido del nombre de la voz guardada que quiere clonar. 2. El sistema detecta el comando. 3. El sistema busca las voces guardadas por el usuario. 4. El sistema envía un mensaje indicando que se va a clonar la voz de la persona elegida.
Secuencia alternativa	1.1.a. El usuario envía un mensaje de texto que no es ningún comando predeterminado ni ninguna custom. 1.1.b. El sistema envía un mensaje diciendo informando que el comando introducido es incorrecto y que vuelva a enviar /customvoices si el usuario quiere seleccionar una voz. 1.2.a. El usuario envía un mensaje de audio. 1.2.b. El sistema envía un mensaje indicando que no tiene que enviar ningún audio, sino enviar un comando.
Postcondiciones	1. El sistema ha enviado un mensaje por el chat. 2. El valor de NAME ha cambiado al elegido por el usuario. 3. El valor de WAITING_FOR_CUSTOM ha cambiado a FALSE.

Cuadro 4.17: CU-15: Seleccionar custom

Identificador	**CU-16: Clonar voz**
Actor principal	Usuario
Actor implicado	API Telegram API Tortoise API Google
Descripción	El usuario envía un mensaje y el sistema clona la voz de una persona con el contenido del mensaje
Precondiciones	1. El usuario ha iniciado un chat con el bot 2. El bot está en cloningmode 3. Hay una voz seleccionada
Secuencia principal	1. El usuario envía un mensaje con el texto que quiere clonar. 2. El sistema recibe el texto enviado. 3. El sistema envía un mensaje de texto indicando que se está generando el audio. 4. El sistema genera un fichero de audio con la voz generada y lo almacena en el sistema. 5. El sistema envía un mensaje de texto indicando que el audio generado es el siguiente. 6. El sistema envía un mensaje de voz con el contenido del audio generado. 7. El sistema elimina el fichero de audio generado.
Secuencia alternativa	1.1.a. El usuario envía un mensaje de audio. 1.1.b. El sistema detecta el contenido del mensaje. 1.1.c. El sistema envía un mensaje con el texto detectado en el mensaje anterior y pasa al punto 3 de la secuencia principal.
Postcondiciones	1. El sistema ha enviado varios mensajes de texto por el chat. 2. El sistema ha generado un fichero de audio con la voz clonada. 3. El sistema ha enviado un mensaje de voz.

Cuadro 4.18: CU-16: Clonar voz

Identificador	**CU-17: Guardar custom**
Actor principal	Usuario
Actor implicado	API Telegram
Descripción	El usuario envía un mensaje de voz y el sistema lo guarda para ser usado más adelante para ser clonado
Precondiciones	1. El usuario ha iniciado un chat con el bot 2. El bot está en uploadingmode
Secuencia principal	1. El usuario envía un mensaje de audio o de voz. 2. El sistema recibe el audio enviado. 3. El sistema envía un mensaje de texto indicando que se está procesando el audio. 4. El sistema convierte el audio a formato WAV. 5. El sistema almacena el audio en ficheros de diez segundos de duración. 6. El sistema envía un mensaje indicando que se ha guardado correctamente. 7. El sistema solicita al usuario que introduzca un nombre con el que guardar el audio. 8. El usuario envía un mensaje con dicho nombre. 9. El sistema guarda la voz con el nombre enviado por el usuario. 10. El sistema envía un mensaje indicando que el nombre se ha registrado correctamente.
Secuencia alternativa	4.1.a. El mensaje de voz enviado excede la capacidad máxima de 50 MB. 4.1.b. El sistema envía un mensaje indicando el error. 8.1.a. El nombre ya existe. 8.1.b. El sistema envía un mensaje de texto informando del error y pasa al punto 7.
Postcondiciones	1. El sistema ha enviado varios mensajes de texto por el chat. 2. El sistema ha guardado ficheros de audio con el nombre indicado por el usuario.

Cuadro 4.19: CU-17: Guardar custom

4.4 Diagramas de actividad

Los diagramas de actividad, también conocidos como diagramas de flujo, en el contexto del Lenguaje Unificado de Modelado (UML), son representaciones gráficas del flujo de trabajo [27]. Muestran, de forma esquemática, los pasos que sigue un algoritmo o proceso de un programa desarrollado.

A continuación se muestran los diagramas de actividades de los distintos casos de uso mostrados en el apartado anterior.

Para un mejor entendimiento de los diagramas, conviene aclarar que el actor *Sistema* hace referencia al código en Python que se encarga de gestionar las peticiones del usuario y conectar dichas peticiones con las distintas APIs.

4.4.1 Diagrama de actividad de start, help y commands

Estos tres casos de uso comparten el mismo diagrama de actividad. En ambos, el usuario envía un mensaje de texto con uno de los tres comandos (/start, /help o /commands), el sistema detecta qué comando es, y se envía por el chat un mensaje informativo en función del comando introducido.

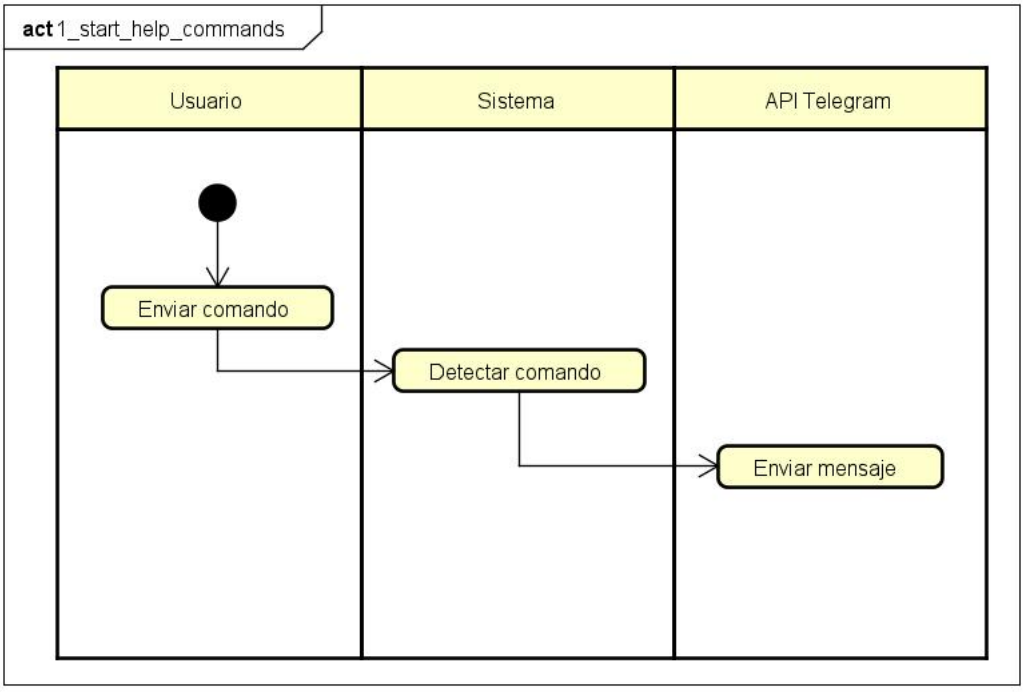


Figura 4.3: Diagrama de actividad de start, help y commands

4.4.2 Diagrama de actividad de cloningmode

En este caso de uso, el usuario envía el mensaje /cloningmode, el sistema se actualiza a modo clonación, y la API de Telegram envía un mensaje informativo por el chat.

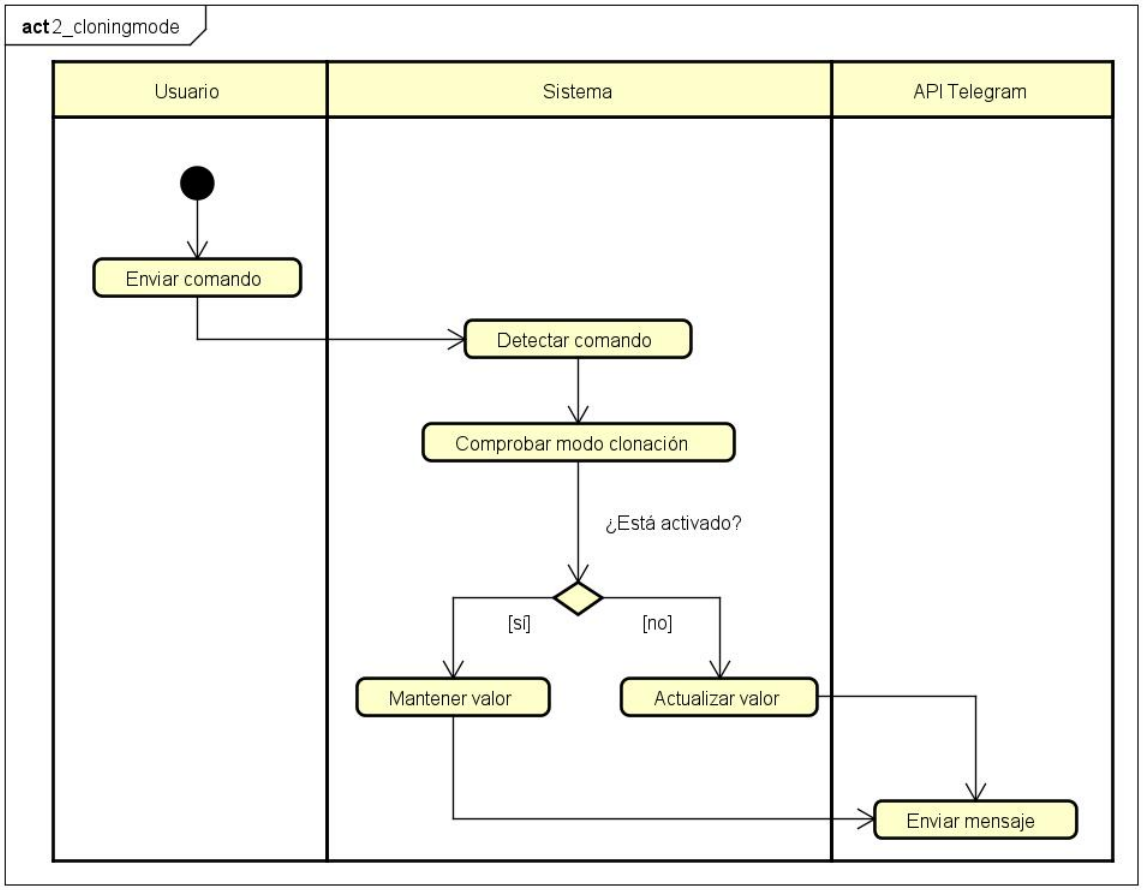


Figura 4.4: Diagrama de actividad de cloningmode

4.4.3 Diagrama de actividad de uploadingmode

Similar al caso de uso anterior, salvo que en este caso el sistema pasa a modo subida.

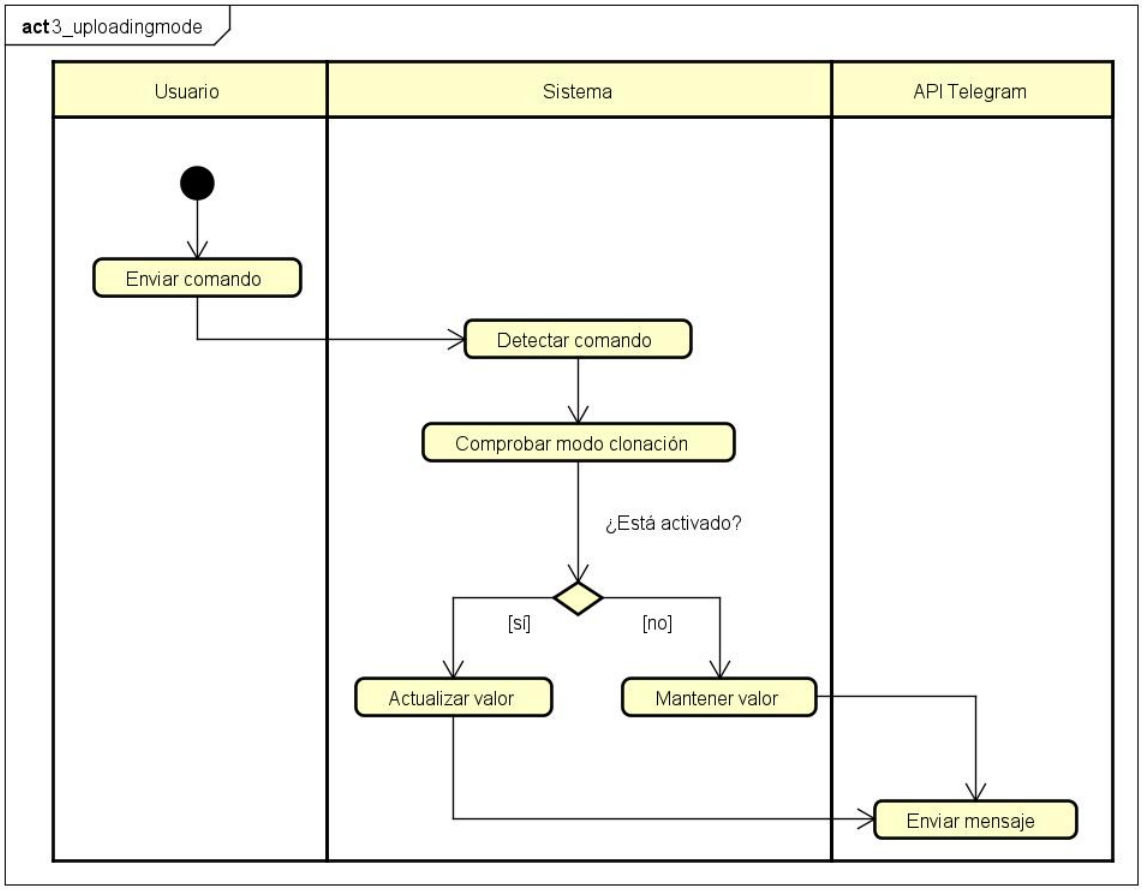


Figura 4.5: Diagrama de actividad de uploadingmode

4.4.4 Diagrama de actividad de ultrafast, fast, standard y highquality

En estos cuatro casos de uso, el usuario envía uno de los cuatro comandos (/ultrafast, /fast, /standard o /highquality) y el sistema actualiza el valor correspondiente del modo de clonación. Posteriormete, se envía un mensaje por chat al usuario informando del cambio.

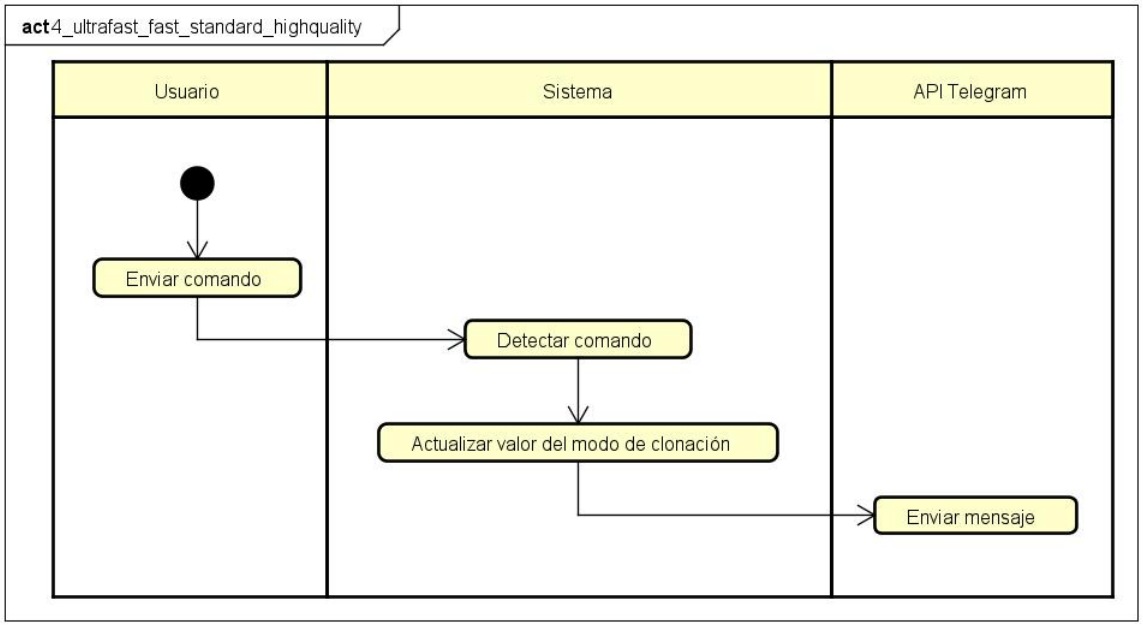


Figura 4.6: Diagrama de actividad de ultrafast, fast, standard y highquality

4.4.5 Diagrama de actividad de cristino y deniro

En estos dos casos de uso, el usuario envía /cristiano o /deniro, y el sistema reemplaza el valor del nombre de la persona a clonar por el correspondiente. A continuación, se informa al usuario del cambio introducido a través de un mensaje por el chat.

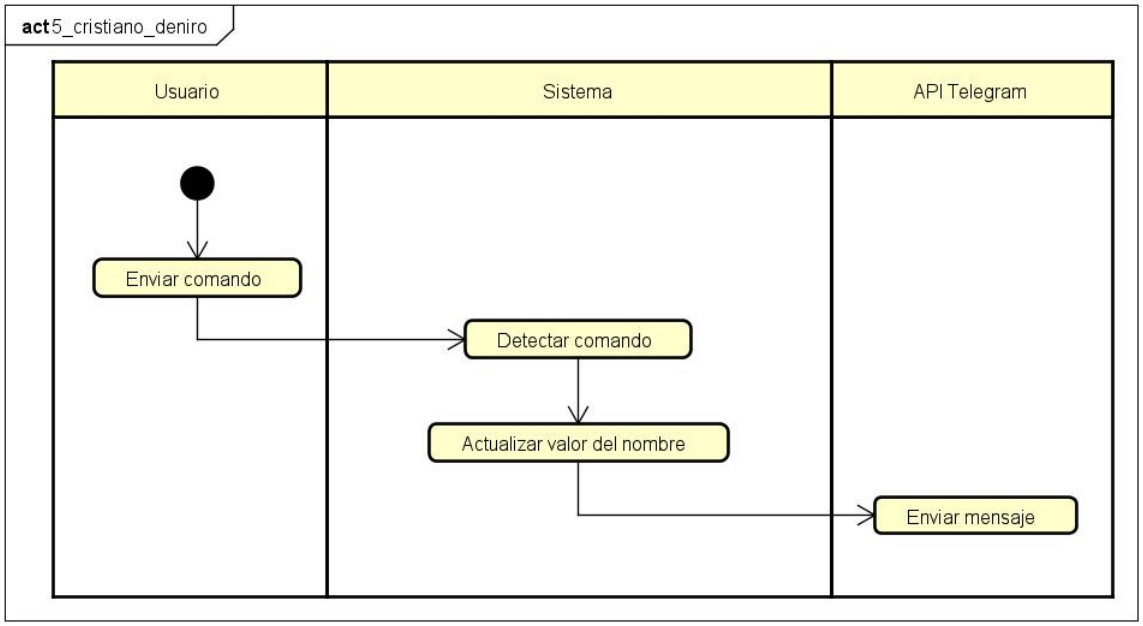


Figura 4.7: Diagrama de actividad de cristiano y deniro

4.4.6 Diagrama de actividad de customvoices y seleccionar custom

En estos casos de uso, hechos de forma conjunta ya que van de la mano, el usuario envía el comando /customvoices por el chat. A continuación, el sistema detecta el comando y busca las customs guardadas previamente por el usuario y crea un mensaje con ellas, que posteriorme es enviado por chat.

Después, el usuario envía una custom en modo comando (por ejemplo, /estaesmicustom), a lo que el sistema procede a seleccionarla para utilizar dicha voz para la clonación y le informa de ello a través de un mensaje de texto.

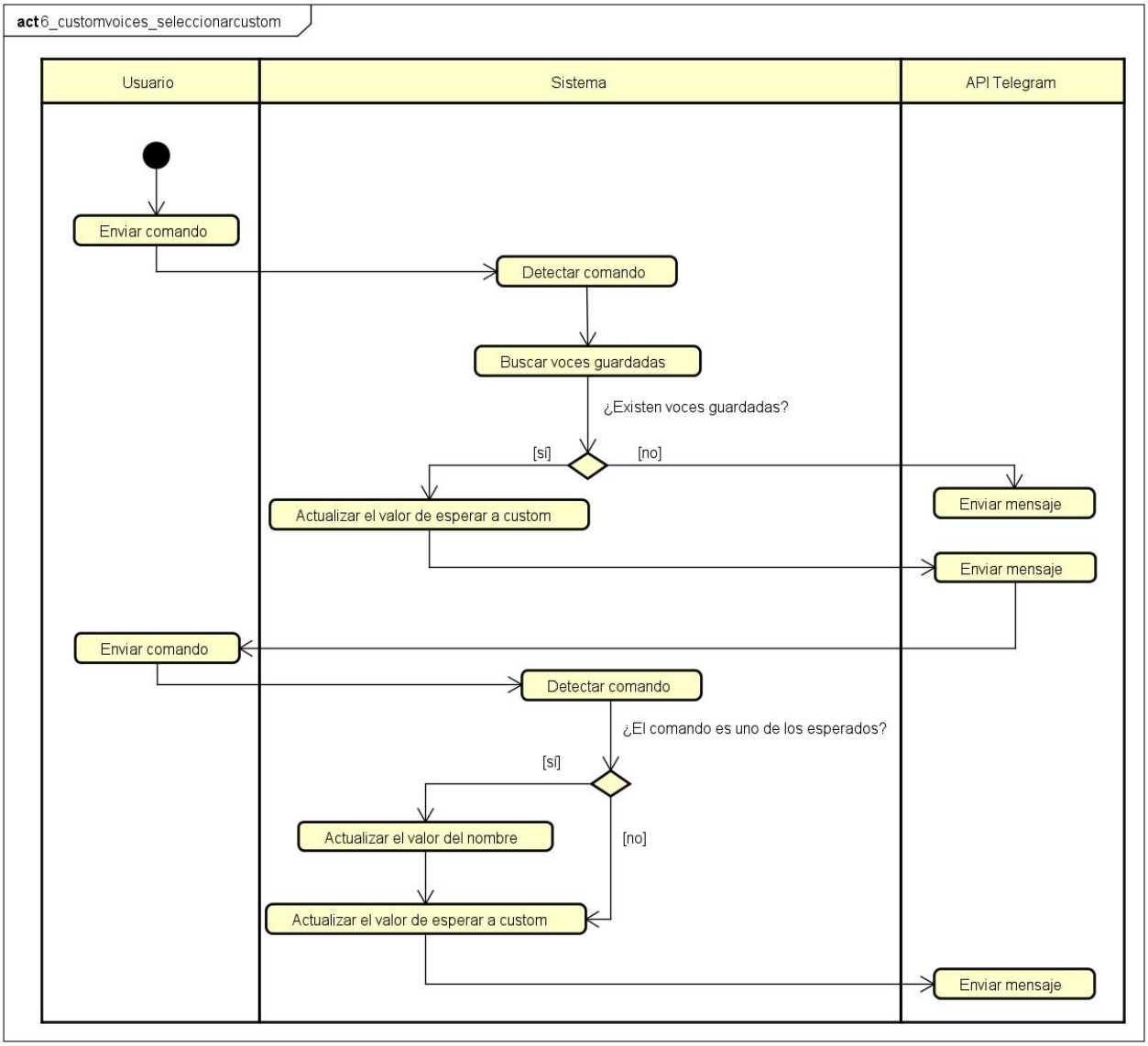


Figura 4.8: Diagrama de actividad de customvoices y seleccionar custom

4.4.7 Diagrama de actividad de deletevoice y borrar custom

En estos casos de uso, igualmente hechos de forma conjunta, el usuario envía el comando /deletevoice por el chat. A continuación, el sistema detecta el comando y busca las customs guardadas previamente por el usuario y crea un mensaje con ellas, que posteriorme es enviado por chat.

Después, el usuario envía una custom en modo comando (por ejemplo, /estaesmicustom), a lo que el sistema procede a eliminarla y le informa de ello a través de un mensaje de texto.

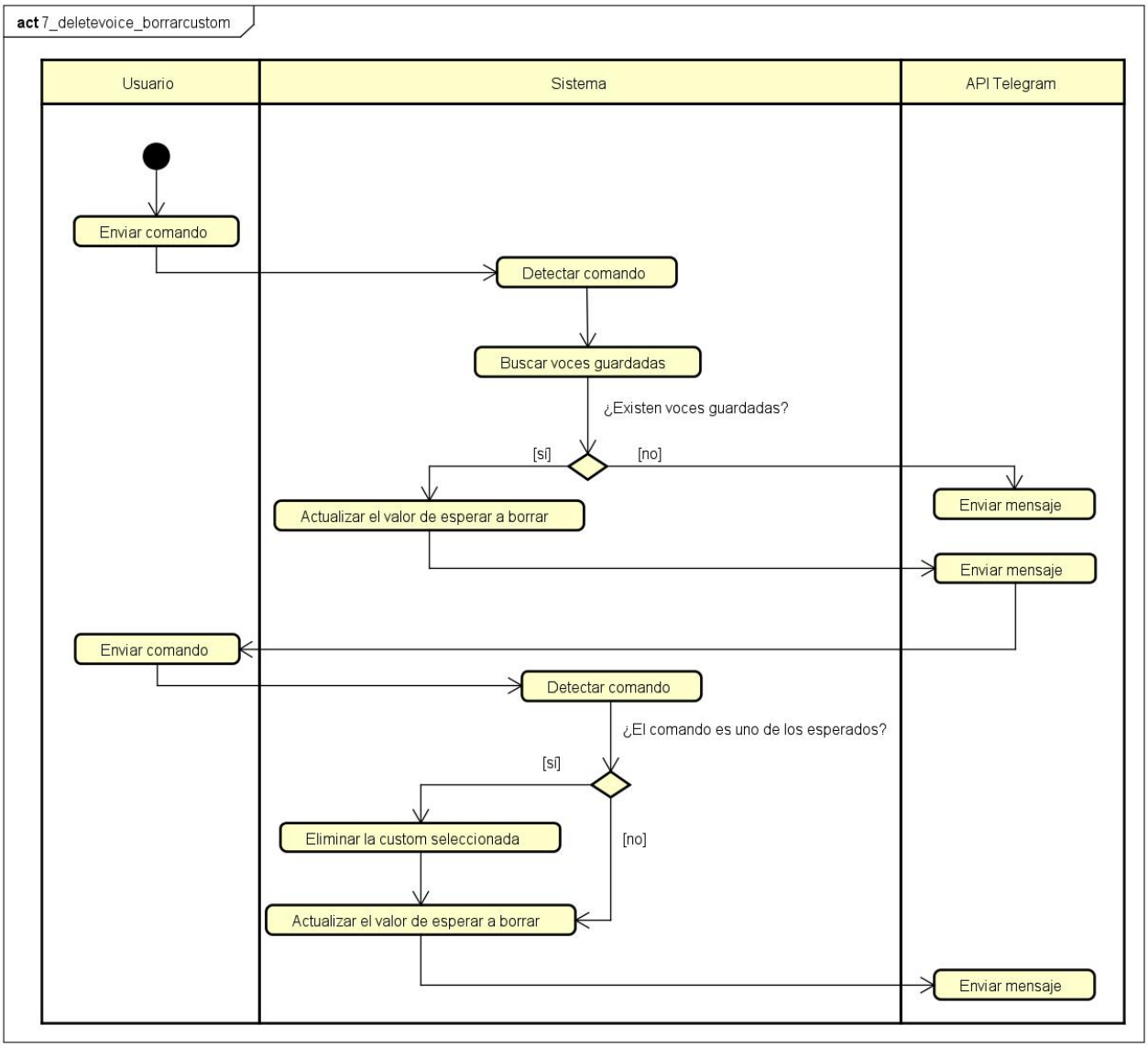


Figura 4.9: Diagrama de actividad de deletevoice y borrar custom

4.4.8 Diagrama de actividad de clonar voz

En este caso de uso, el usuario envía un mensaje de texto o voz por el chat; si es de audio, se extrae el texto. A continuación, se procede a clonar la voz de la persona que esté seleccionada y se envía al usuario por el chat.

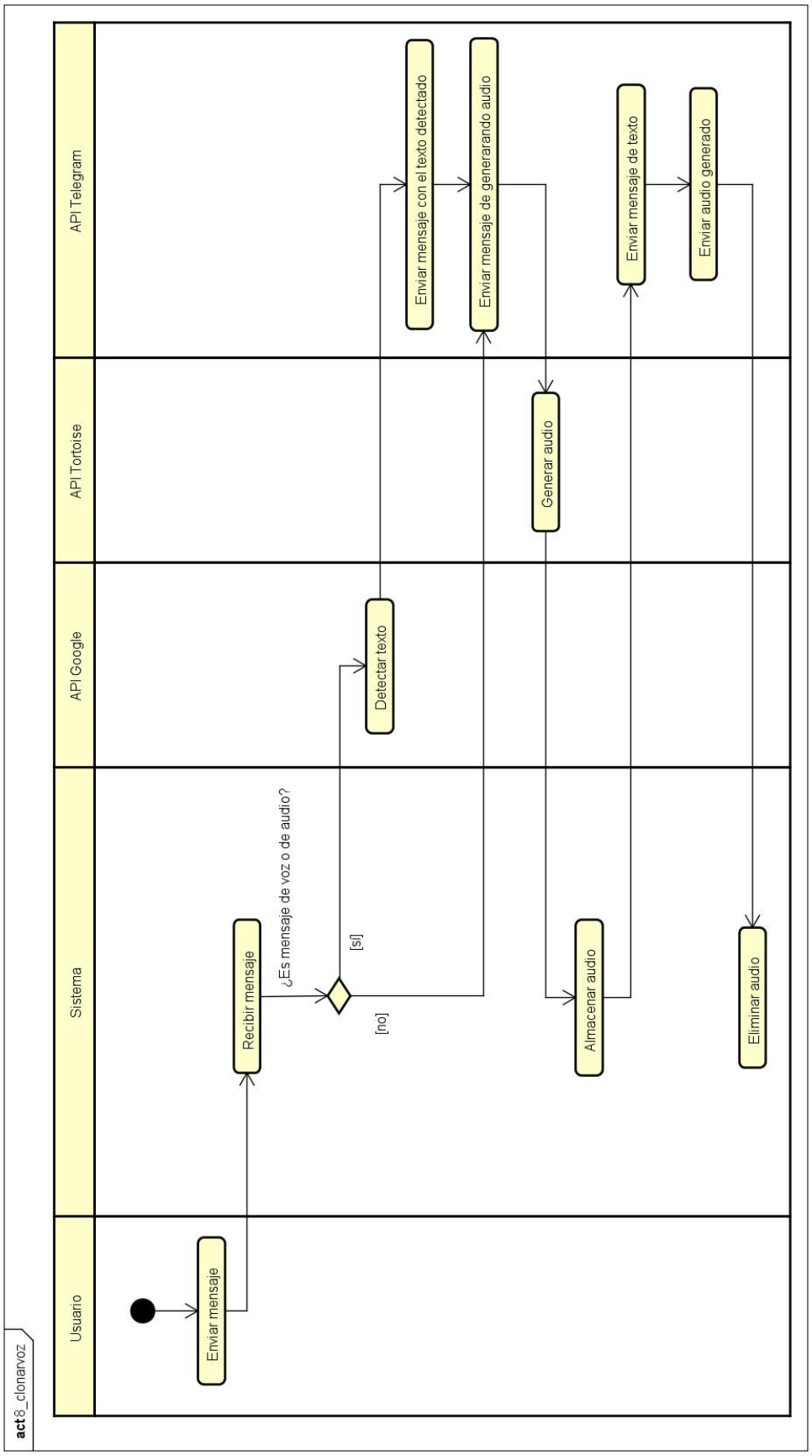


Figura 4.10: Diagrama de actividad de clonar voz

Capítulo 5

Diseño

5.1 Patrón de diseño

El patrón de diseño elegido para el bot es el de cadena de responsabilidad [25]. Este se configura como un enfoque de comportamiento que busca reducir el acoplamiento entre el emisor y el receptor de una solicitud. Para lograr esto, se establece una cadena de objetos receptores, permitiendo que múltiples objetos tengan la capacidad de responder a una solicitud.

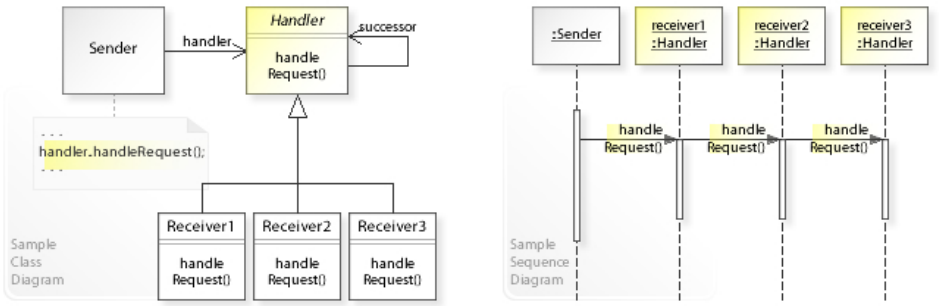


Figura 5.1: Patrón de diseño cadena de responsabilidad [25]

Al enviar un mensaje, el contenedor de nodos envia el mensaje al nodo correspondiente, devolviendo el resultado si la clave coincide o pasando el mensaje al siguiente nodo si no hay coincidencia. Se debe tener en cuenta que una cadena extensa con muchos elementos puede acumular una cantidad significativa de llamadas a procedimientos en memoria, lo que conlleva costos asociados.

Para solucionar ese problema, surgen los *dispatchers*, que pueden enviar comandos en diversas direcciones formando un árbol de responsabilidad.

La biblioteca *python-telegram-bot* [10] permite una implementación sencilla de este patrón, al contener los handlers necesarios para cada tipo de mensaje.

Capítulo 6

Implementación

6.1 Herramientas de Desarrollo

6.1.1 Lenguaje de programación: Python

Python es un lenguaje de programación de alto nivel, interpretado y de propósito general, creado por Guido van Rossum a finales de la década de 1980 y principios de la de 1990 [28]. Su diseño se centra en la legibilidad del código y la productividad del programador, lo que lo convierte en una opción popular tanto para principiantes como para desarrolladores experimentados. Python es conocido por su sintaxis clara y concisa, facilitando la escritura y comprensión del código.

Figura 6.1: Logo de Python [28]

La filosofía de Python destaca principios como la legibilidad cuenta y la simplicidad, fomentando un enfoque práctico y eficiente en el desarrollo de software. La versatilidad de Python se refleja en su capacidad para adaptarse a diversos contextos, desde desarrollo web y análisis de datos hasta inteligencia artificial y aprendizaje automático. Además, su naturaleza de código abierto ha contribuido a una comunidad activa que desarrolla bibliotecas y marcos de trabajo, fortaleciendo aún más su posición como una herramienta poderosa y accesible en el mundo de la programación.

En resumen, Python es un lenguaje que combina simplicidad, legibilidad y versatilidad, convirtiéndolo en una opción preferida para una amplia gama de aplicaciones y proyectos.

6.1.2 Entorno de desarrollo: Visual Studio Code

Visual Studio Code (VS Code) es un entorno de desarrollo integrado (IDE) altamente personalizable desarrollado por Microsoft. Está diseñado para ser eficiente y fácil de usar, y se ha vuelto

extremadamente popular entre los desarrolladores de software. Su interfaz intuitiva y su amplia gama de extensiones contribuyen a una experiencia de desarrollo fluida.

Figura 6.2: Logo de VS Code [24]

6.1.3 Redacción de la memoria: Overleaf

Overleaf es un editor online de textos en formato LaTeXque sirve para escribir documentos, especialmente en el ámbito científico. Destaca por las facilidades que otorga a la hora de colaborar varias personas en la escritura de dichos documentos.

Figura 6.3: Logo de Overleaf [24]

6.2 Implementación

En esta sección se explica de forma detallada la implementación del bot, desde la creación del mismo hasta su programación para realizar su función de clonador.

6.2.1 Preparación de TorToiSe

Para la realización de este trabajo, lo primero de todo fue la puesta a punto de TorToiSe para su funcionamiento en el entorno local. Para ello, se clonó el repositorio de GitHub creado por James Betker [9] y posteriormente se instalaron las bibliotecas necesarias siguiendo las indicaciones incluidas en el fichero *README.md*.

6.2.2 Creación del bot

Para la creación del bot se ha usado otro bot de Telegram, llamado *BotFather* [4], que desempeña el papel de *padre* de todos los bots en la plataforma. Se trata de un bot especial creado por los desarrolladores de Telegram que permite a un usuario crear y gestionar sus propios bots de Telegram de manera sencilla. El usuarios puede interactuar con *BotFather* para obtener un token único que identifique a su bot, lo que le permite configurar y personalizar las funciones del bot según sus necesidades.

Figura 6.4: Logo de *BotFather* [23]

Una de las características destacadas del *BotFather* es su interfaz intuitiva que guía al usuario a través del proceso de creación de un nuevo bot, proporcionando opciones para establecer el nombre del bot, su descripción y otras configuraciones importantes. Además, *BotFather* ofrece comandos adicionales para personalizar aún más el comportamiento del bot, como la adición de comandos personalizados, la configuración de imágenes de perfil y la asignación de permisos específicos.

Para crear el bot, se introdujo el comando */newbot*, y *BotFather* devolvió lo siguiente:

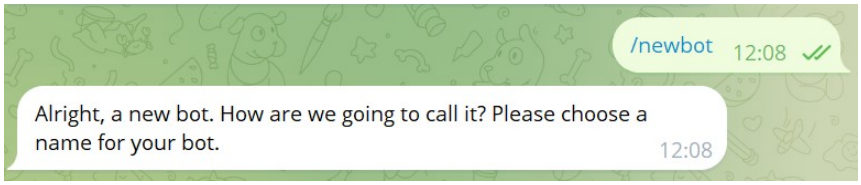


Figura 6.5: Creación del bot

Tras introducir *Voice Cloner*, *BotFather* envió a continuación lo siguiente para introducir el nombre de usuario:

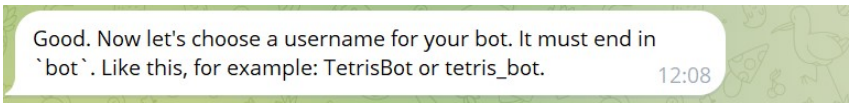


Figura 6.6: Nombre de usuario del bot

Se introdujo *@tfg_informatica_bot*, y con ello el bot fue creado.

A continuación, se introdujo una descripción mediante el comando */setdescription*:

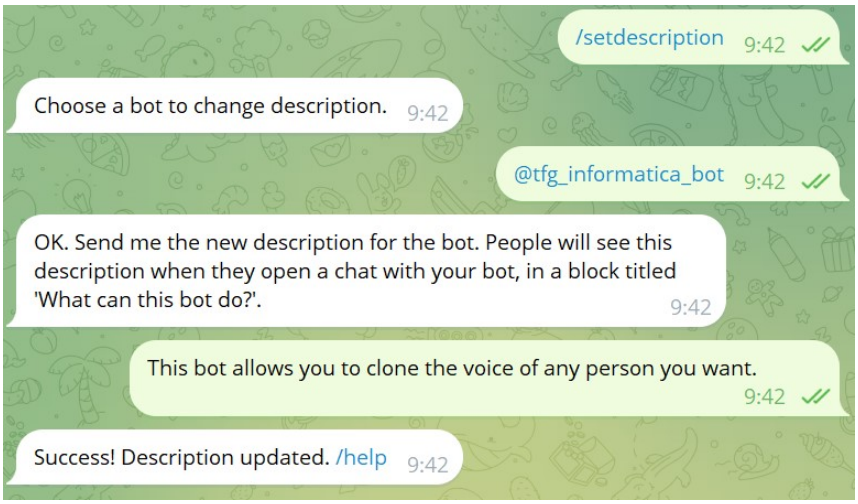


Figura 6.7: Descripción del bot

Finalmente, a partir del comando */setcommands*, se introdujo una lista de todos los comandos disponibles con una breve descripción de cada uno de ellos:

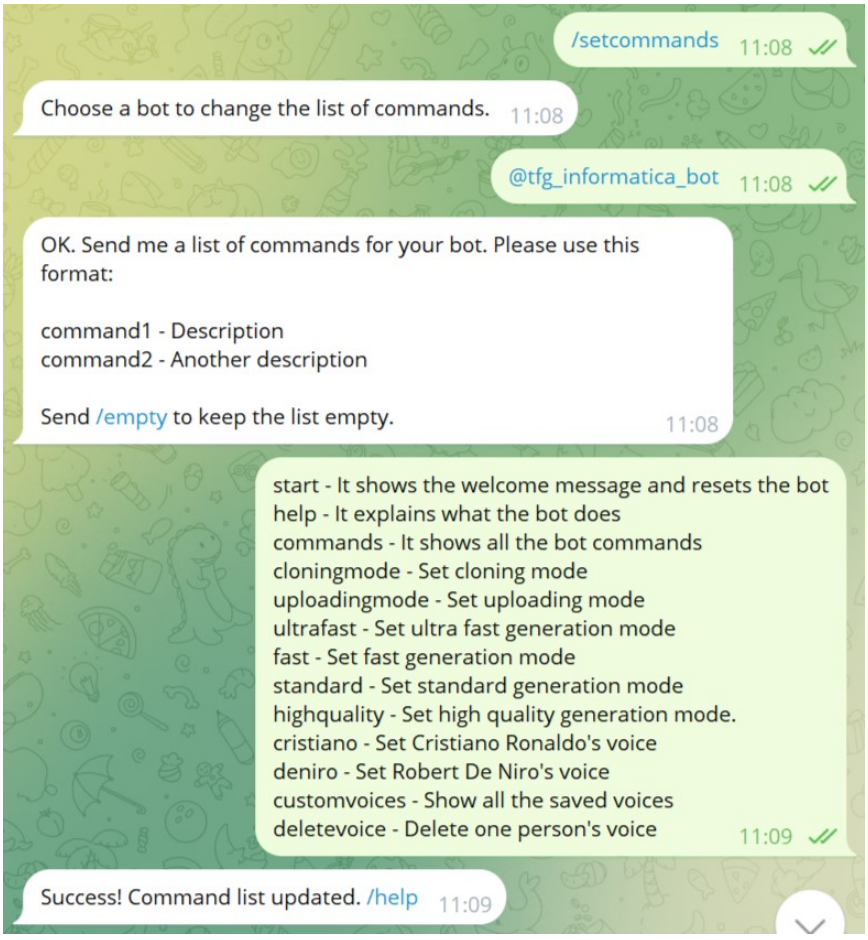


Figura 6.8: Comandos del bot

6.2.3 Implementación del bot

En este apartado se explica la implementación de código usado para programar el bot. Como se ha indicado anteriormente, el lenguaje de programación utilizado fue Python.

Todo el código del bot se desarrolló en un único fichero llamado *voice_cloner.py*, que se creó en el directorio raíz del repositorio clonado explicado en 6.2.1.

Bibliotecas

Para sincronizar un bot con Python, el propio lenguaje tiene una biblioteca llamada python-telegram-bot (telegram en 6.9), que facilita mucho dicha sincronización. Además de esa biblioteca, también se utilizaron las siguientes:

```
1  import math
2  import os
3  import shutil
4  import speech_recognition as sr
5  import torchaudio
6  from pydub import AudioSegment
7  from telegram import Update
8  from telegram.ext import Application, CommandHandler, ContextTypes, filters, MessageHandler
9  from tortoise import api, utils
10 from typing import Final
11
```

Figura 6.9: Bibliotecas utilizadas

- *math* [17]: utilizada para realizar diversas operaciones matemáticas.
- *os* [18], *shutil* [19]: utilizada para realizar operaciones en ficheros y carpetas.
- *speech_recognition* [14]: utilizada para convertir voz en texto.
- *torchaudio* [15]: utilizada para guardar los audios generados.
- *pydub* [13]: utilizada para trabajar con ficheros *WAV*.
- *telegram*: utilizada para la conexión con el bot de Telegram.
- *tortoise*: uilizada para la clonación de voz.
- *typing* [20]: utilizada para crear constantes.

Conexión con Telegram y handlers

Lo primero de todo fue conectar el bot de Telegram con el código [7]. Con la biblioteca *telegram* mencionada anteriormente, y mediante un token único del bot, se realizó dicha conexión.

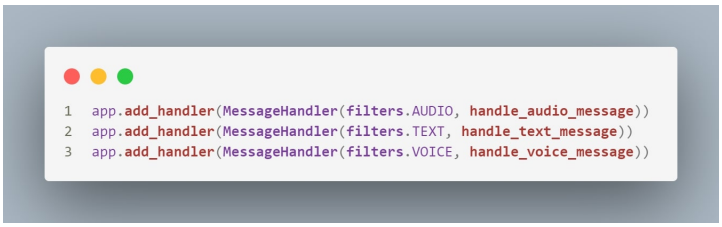
```
app = Application.builder().token(TOKEN).build()
```

Figura 6.10: Conexión con el bot

A continuación, se añadieron handlers que se encargan de manejar las distintas situaciones a las que se puede enfrentar el bot (comandos, mensajes de audio, voz y texto, y para cuando haya errores).

```
# Default
app.add_handler(CommandHandler('start', start_command))
app.add_handler(CommandHandler('help', help_command))
app.add_handler(CommandHandler('commands', commands_command))
# Normal
app.add_handler(CommandHandler('cloningmode', cloningmode_command))
app.add_handler(CommandHandler('uploadingmode', uploadingmode_command))
# Mode
app.add_handler(CommandHandler('ultrafast', ultrafast_command))
app.add_handler(CommandHandler('fast', fast_command))
app.add_handler(CommandHandler('standard', standard_command))
app.add_handler(CommandHandler('highquality', highquality_command))
# Voices
app.add_handler(CommandHandler('cristiano', cristiano_command))
app.add_handler(CommandHandler('deniro', deniro_command))
# Misc
app.add_handler(CommandHandler('customvoices', customvoices_command))
app.add_handler(CommandHandler('deletevoice', deletecustom_command))
```

Figura 6.11: Handlers de comandos



```
app.add_handler(MessageHandler(filters.AUDIO, handle_audio_message))
app.add_handler(MessageHandler(filters.TEXT, handle_text_message))
app.add_handler(MessageHandler(filters.VOICE, handle_voice_message))
```

Figura 6.12: Handlers de tipo de mensaje

```
app.add_error_handler(error)
```

Figura 6.13: Handler de error

Conexión con TorToiSe

El siguiente paso fue integrar TorToiSe con el bot. Para ello, se realizó una clase, llamada *VoiceCloner*, que conecta con la API de TorToiSe para realizar la clonación de voz. Además, esta clase cuenta con una función llamada *synthesize_voice*, que realiza la síntesis de voz y guarda el fichero wav generado en el sistema para su posterior envío por el chat.

```python
class VoiceCloner:

    def __init__(self):
        self.tts = api.TextToSpeech(use_deepspeed=True, kv_cache=True)

    def synthesize_voice(self, preset, voice, text, chat_id):
        voice_samples, conditioning_latents = utils.audio.load_voice(voice)
        gen = self.tts.tts_with_preset(text, voice_samples=voice_samples, conditioning_latents=conditioning_latents,
                                       preset=preset)
        path = f'tortoise/voices/user_{chat_id}_tmp'
        create_path(path)
        path = f'{path}/generated.wav'
        torchaudio.save(path, gen.squeeze(0).cpu(), 24000)
        return path
```

Figura 6.14: Clase VoiceCloner

La API proporciona una clase llamada *TextToSpeech*, que es el principal punto de partida de TorToise. Esta contiene a su vez una función con nombre *tts_with_preset* que recibe, entre otros parámetros, las muestras de voz y un texto y se encarga de generar la voz clonada, siguiendo los pasos descritos en 3.7.

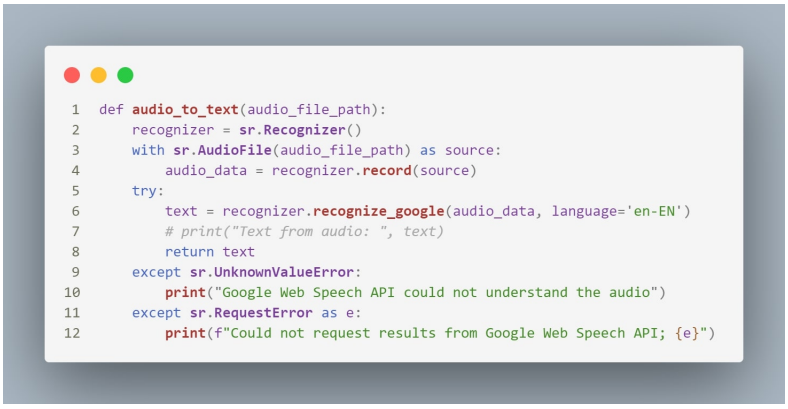
Clonación

El caso más sencillo de clonación es el de clonar un mensaje de texto; para ello se utiliza la función *handle_text_message*, que obtiene primero el mensaje de texto, después envía un mensaje al usuario indicándole que se está generando el audio, y finalmente envía el audio con la voz clonada y elimina el fichero generado.

```
text: str = update.message.text
chat_id = update.message.chat.id

# print(f'User ({chat_id}) in {message_type}: "{text}"')
message_to_delete = await update.message.reply_text('Generating audio, please wait...')

voice_cloned_path = CLONER.synthesize_voice(MODE, NAME, text, chat_id)

await context.bot.deleteMessage(message_id = message_to_delete.message_id, chat_id = update.message.chat_id)
await update.message.reply_text('This is the generated audio:')
await update.message.reply_voice(voice_cloned_path)
os.remove(voice_cloned_path)
```

Figura 6.15: Clonar un mensaje de texto

El otro caso de clonación (por mensaje de audio o de voz) requiere de un procesamiento un poco más complejo, ya que primero hay que convertir lo dicho por una persona a texto (con la biblioteca *speech_recognition*).



```
def audio_to_text(audio_file_path):
    recognizer = sr.Recognizer()
    with sr.AudioFile(audio_file_path) as source:
        audio_data = recognizer.record(source)
    try:
        text = recognizer.recognize_google(audio_data, language='en-EN')
        # print("Text from audio: ", text)
        return text
    except sr.UnknownValueError:
        print("Google Web Speech API could not understand the audio")
    except sr.RequestError as e:
        print(f"Could not request results from Google Web Speech API; {e}")
```

Figura 6.16: Detectar texto en audio

Posteriormente se realiza lo mismo que en el caso de mensaje de texto. Además, hay que comprobar que el fichero de audio enviado esté en formato *WAV*, ya que el bot ha sido diseñado para soportar solo ese formato de audio. En caso de que sea un mensaje de voz, se realiza la conversión a *WAV*.

```python
try:
    filepath_wav = f'{path}/{message_id}.wav'
    await file.download_to_drive(filepath_wav)

    text = audio_to_text(filepath_wav)

    await context.bot.deleteMessage(message_id = message_to_delete.message_id, chat_id = update.message.chat_id)
    message_to_delete = await update.message.reply_text(f'You have said "{text}".')
    message_to_delete_2 = await update.message.reply_text('Generating audio, please wait...')

    voice_cloned_path = CLONER.synthesize_voice(MODE, NAME, text, chat_id)

    await context.bot.deleteMessage(message_id = message_to_delete_2.message_id, chat_id = update.message.chat_id)
    await update.message.reply_text('This is the generated audio:')
    await update.message.reply_voice(voice_cloned_path)
    os.remove(filepath_wav)
    os.remove(voice_cloned_path)
except:
    message = 'Error processing the audio. Please, send an audio in WAV format.'
    await update.message.reply_text(message)
```

Figura 6.17: Clonar un mensaje de audio

Guardar voz

Para guardar un mensaje de voz enviado por el usuario para usarlo como referencia para clonarlo, se utiliza lo siguiente:

```python
file_id = update.message.voice.file_id
file = await context.bot.get_file(file_id)
message_to_delete = await update.message.reply_text('Processing audio, please wait...')

chat_id = update.message.chat_id
message_id = update.message.message_id
path = f'tortoise/voices/tmp_user/user_{chat_id}_tmp'
create_path(path)
filepath_ogg = f'{path}/{message_id}.ogg'
await file.download_to_drive(filepath_ogg)
# If the audio size is greater than 50 MB.
if get_folder_size(path) >= 50:
    os.remove(filepath_ogg)
    message = 'The audio you have sent exceeds the maximum capacity of 50 MB.'
    await context.bot.deleteMessage(message_id = message_to_delete.message_id, chat_id = update.message.chat_id)
    await update.message.reply_text(message)
else:
    filepath_wav = f'{path}/{message_id}.wav'
    ogg_to_wav(filepath_ogg, filepath_wav)
    voices = os.listdir('tortoise/voices')
    total_size = 0
    for voice in voices:
        if voice.startswith(f'user_{chat_id}_saved'):
            total_size += get_folder_size('tortoise/voices/' + voice)
    # If all the custom voices sizes are greater than 50 MB.
    if total_size >= 50:
        message = 'You have exceed the maximum capacity of 50 MB.'
        await context.bot.deleteMessage(message_id = message_to_delete.message_id, chat_id = update.message.chat_id)
        await update.message.reply_text(message)
    else:
        split_wav(filepath_wav, chat_id, message_id)
        await update.message.reply_text(f'Audio processed correctly.\nYou have used {round(total_size, 2)}/50 MB.')
        await update.message.reply_text("Please, choose a name for the person's voice (spaces will be deleted).")
    await context.bot.deleteMessage(message_id = message_to_delete.message_id, chat_id = update.message.chat_id)
    os.remove(filepath_ogg)
    os.remove(filepath_wav)
    MUST_BE_SAVED = True
```

Figura 6.18: Guardar un mensaje de voz

Tras procesar el mensaje, se comprueba que no se exceda la capacidad límite de 50 MB establecida para cada usuario. A continuación, se convierte a un fichero *WAV* y se vuelve a realizar la comprobación de que no exceda la capacidad límite. Finalmente, si todo ha ido bien, se divide en fragmentos de diez segundos, se le informa al usuario con la cantidad de almacenamiento que ha utilizado, y se le pide que introduzca un nombre con el que guardar el audio para utilizarlo posteriormente para clonar.

Las voces guardadas se almacenan de la siguiente manera: dentro de la ruta *tortoise/voices*, cuando se guarda una voz, se generan un directorio, el cual tiene la estructura *user_idChat_saved_voz* (siendo *voz* el nombre elegido por el usuario), donde se almacenan los ficheros *WAV* divididos. De esta forma, es sencillo recuperar la lista de voces guardadas de un mismo usuario. Esto a su vez hace que, cuando un usuario elimine la conversación e inicie una nueva, ya no pueda acceder a las voces guardadas previamente, ya que están relacionadas con el identificador del chat, no con el del usuario.

Otras funciones

Además de las funciones vistas anteriormente, destacan otras sin las cuales el bot no funcinaría.

Los mensajes de audio, cuando se extraen, se almacenan en formato *Ogg*. TorToiSe requiere que los ficheros sean en formato *WAV*, por lo que había que convertirlos de algún modo.

```python
def ogg_to_wav(filepath_ogg: str, filepath_wav: str):
    sound = AudioSegment.from_ogg(filepath_ogg)
    sound.export(filepath_wav, format='wav')
```

Figura 6.19: *Ogg* a *WAV*

Para ello, se hace uso de la biblioteca *pydub*, que mediante la función *from_ogg* de *AudioSegment* permite extraer la información de un fichero *Ogg* para posteriormente exportarla en formato *WAV*.

Además, TorToiSe procesa fragmentos de audio de unos diez segundos de duración, por lo que los ficheros de audio con una duración mayor tenían que ser divididos. Para ello, se creó la función *split_wav*, que se muestra a continuación.

```python
def split_wav(filepath: str, chat_id: str, message_id: str):
    global PATH_TO_DELETE_OR_RENAME
    sec = 10
    input_audio_file = filepath
    speech = AudioSegment.from_wav(input_audio_file)
    batch_size = sec * 1000
    duracion = speech.duration_seconds
    batches = math.ceil(duracion / sec)
    path = f'tortoise/voices/user_{chat_id}_saved_{message_id}'
    if not os.path.exists(path):
        os.makedirs(path)
    PATH_TO_DELETE_OR_RENAME = path
    inicio = 0
    for i in range(batches):
        pedazo = speech[inicio: inicio + batch_size]
        pedazo.export(f'{path}/{i+1}.wav', format='wav')
        inicio+= batch_size
```

Figura 6.20: *split_wav*

Capítulo 7

Pruebas

Para comprobar el correcto funcionamiento de los distintos casos de uso, se realizaron los siguientes casos de prueba al acabar el desarrollo del código, dando así por finalizado el desarrollo así como el trabajo:

Identificador	CP-01
Nombre	CU-01: start
Entrada	Mensaje de texto, con contenido /start
Salida esperada	Mensaje de bienvenida al usuario
Salida obtenida	Salida esperada

Cuadro 7.1: CP-01 - CU-01: start

Identificador	CP-02
Nombre	CU-02: help
Entrada	Mensaje de texto, con contenido /help
Salida esperada	Mensaje de cómo se utiliza el bot
Salida obtenida	Salida esperada

Cuadro 7.2: CP-02 - CU-02: help

Identificador	CP-03
Nombre	CU-03: commands
Entrada	Mensaje de texto, con contenido /commands
Salida esperada	Mensaje con la lista de comandos
Salida obtenida	Salida esperada

Cuadro 7.3: CP-03 - CU-03: commands

Identificador	CP-04
Nombre	CU-04: cloningmode
Entrada	Mensaje de texto, con contenido /cloningmode
Salida esperada	Mensaje indicando que se ha cambiado a modo clonación
Salida obtenida	Salida esperada

Cuadro 7.4: CP-04 - CU-04: cloningmode

Identificador	CP-05
Nombre	CU-05: uploadingmode
Entrada	Mensaje de texto, con contenido /uploadingmode
Salida esperada	Mensaje indicando que se ha cambiado a modo subida
Salida obtenida	Salida esperada

Cuadro 7.5: CP-05 - CU-05: uploadingmode

Identificador	CP-06
Nombre	CU-06-09: quality
Entrada	Cuatro mensajes de texto, con contenido /ultrafast, /fast, /standard y /highquality
Salida esperada	Cuatro mensajes de texto, indicando que el modo de clonación se ha cambiado al indicado en la entrada
Salida obtenida	Salida esperada

Cuadro 7.6: CP-06: quality

Identificador	CP-07
Nombre	CU-10-11: voice
Entrada	Dos mensajes de texto, con contenido /cristiano y /deniro
Salida esperada	Dos mensajes de texto, indicando que la voz clonada va a ser la de Cristiano o De Niro
Salida obtenida	Salida esperada

Cuadro 7.7: CP-07 - CU-10-11: voice

Identificador	CP-08
Nombre	CU-12: customvoices
Entrada	Mensaje de texto, con contenido /customvoices
Salida esperada	Mensaje de texto con una lista de las voces guardadas por el usuario
Salida obtenida	Salida esperada

Cuadro 7.8: CP-08 - CU-12: customvoices

Identificador	CP-09
Nombre	CU-13-14: deletevoice
Entrada	Mensaje de texto, con contenido /deletevoice
Salida esperada	Mensaje de texto con una lista de las voces guardadas por el usuario
Salida obtenida	Salida esperada

Cuadro 7.9: CP-09 - CU-13: deletevoice

Identificador	CP-10
Nombre	CU-15: Seleccionar custom
Entrada	Dos mensajes de texto, primero /customvoices, y a continuación /Isabel
Salida esperada	Mensaje de texto que indique que se va a clonar la voz de Isabel
Salida obtenida	Salida esperada

Cuadro 7.10: CP-10 - CU-15: Seleccionar custom

Identificador	CP-11
Nombre	CU-16: Clonar voz
Entrada	Mensaje de texto, con contenido ’Hello, it’s me’
Salida esperada	Mensaje de voz replicando el texto con la voz de Cristiano Ronaldo
Salida obtenida	Salida esperada

Cuadro 7.11: CP-11 - CU-16: Clonar voz

Se realizaron también pruebas para comprobar que se detectaban bien los errores que no han sido incluidas en este documento para no alargar en exceso esta parte.

Capítulo 8

Conclusiones y líneas futuras

8.1 Conclusiones

A lo largo de este trabajo se ha podido ver el potencial que tienen las tecnologías de hoy en día para replicar algo tan propio de una persona como su voz, y la facilidad con las que esta tecnología puede ser implementada en una aplicación de mensajería instantánea como es Telegram. En este trabajo en concreto, se ha visto tanto el reconocimiento del habla como la síntesis de voz adaptada para obtener un sonido similiar al de una persona ya existente.

Esto trae consigo inherentemente varias preguntas éticas, pero quizá la que mejor englobe todo sea: ¿Hasta qué punto está bien imitar la voz de una persona? Porque en este caso, cuando se escucha detenidamente, sí se puede detectar que la voz está generada de manera artificial pero, ¿qué pasará cuando esta tecnología evolucione y cree voces que no se puedan distinguir de las reales?

Una posible respuesta sería: mientras se use con fines lúdicos o para mejorar la vida de la gente estará bien, el problema vendrá cuando se utilice sin consentimiento o para suplantar la identidad de una persona. Al final, como en casi todos los ámbitos de la vida, hay que respetar la libertad individual de cada persona, para así preservar la integridad de cada uno. El avance tecnológido que presenta la inteligencia artificial es muy potente, que puede generar grandes beneficios bien usada. Sin embargo, debido presisamente a su potencial, es muy llamativa a ojos de aquellos que quieren darle un uso indebido.

8.2 Líneas futuras

Si bien se ha conseguido el objetivo principal de este trabajo (crear un bot en Telegram que permita clonar la voz humana), siempre se pueden realizar mejoras para obtener un mejor funcionamiento y/o rendimiento:

- Mejorar la tecnología de clonación de voz, ya sea mejorando la calidad o la velocidad de la generación de voz.

- Realizar un mejor seguimiento de las acciones que realiza el usuario, para así llevar un mejor control de los errores.
- Almacenar las voces en una base de datos, ya que en este bot se guardan donde esté alojado el código. Esto actualmente no supone ningún problema, pero en el caso en que el bot empezara a tener numerosos usuarios, acabaría ocasionando problemas de almacenamiento.
- Integrar un comando adicional que permita al usuario obtener la información acerca del estado en el que se encuentra (modo subida o clonación, qué voz va a clonar, calidad seleccionada...).
- Permitir el envío de ficheros de audio en otros formatos (por ejemplo *mp3*).

Appendices

Apéndice A

Manual de Instalación

En este apéndice se resumen las instrucciones para instalar el bot en cualquier equipo. Las instrucciones detalladas se podrán encontrar en el repositorio del proyecto en gitlab de la Escuela (https://gitlab.inf.uva.es/javcoch/tortoise-tts).

Para utilizarlo en local, es necesario tener una GPU de NVIDIA. Lo primero de todo será instalar Python desde su página web oficial [16].

A continuación, hay que instalar miniconda [1], y ejecutar los siguientes comandos desde el prompt de anaconda:

- Activación de tortoise:

```
conda create --name tortoise python=3.9 numba inflect
conda activate tortoise
```

- Instalación de las librerías y dependencias:

```
conda install pytorch torchvision torchaudio pytorch-cuda=11.7
    -c pytorch -c nvidia
conda install transformers=4.29.2
```

- Clonación del repositorio:

```
git clone https://gitlab.inf.uva.es/javcoch/tortoise-tts
```

- Configuración de tortoise:

```
cd tortoise-tts
python setup.py install
```

- Lanzamiento del bot:

```
python voicecloner.py
```

Con ello, tendremos el bot ejecutándose en local.

Apéndice B

Manual de Usuario

Para poder utilizar el bot, lo primero de todo es tener una cuenta creada en Telegram. Una vez iniciada sesión, habrá que buscar en la barra de búsqueda *@tfg_informatica_bot* y clickar en el chat que aparezca.

Una vez dentro del chat, se mostrará cierta información correspondiente al bot. Escribiendo */start*, el bot devolverá un mensaje a partir del cual se irán indicando todas las funcionalidades del bot.

El bot tiene dos modos: clonación y subida. En modo clonación, cuando se envíen audios, texto o voces, los devolverá clonados con la voz de la persona correspondiente (por defecto la de Cristiano Ronaldo). Para cambiar entre estos modos, habrá que enviar */cloningmode* o */uploadingmode*.

Si lo que se quiere es clonar un mensaje de texto o de voz, simplemente enviando dicho mensaje (en modo clonación) el bot lo clonará con la voz de Cristiano Ronaldo.

Hay cuatro voces disponibles de forma directa: */cristiano*, */deniro*, */trump* y */psanchez*. Para elegir otra de las voces disponibles por el sistema, primero hay que enviar el comando */voices*. En el mensaje devuelto por el bot, se podrá seleccionar una voz de la lista.

Si lo que se quiere es guardar una voz para posteriormente utilizarla para clonar, habrá que cambiar primero a modo subida enviando */uploadingmode* y a continuación enviando el mensaje de audio o voz que se desee. El bot procesará ese audio y pedirá que se introduzca un nombre con el que guardarlo. Enviando el nombre deseado, este proceso estará listo.

Para utilizar una voz guardada, habrá que enviar el comando */customvoices*. El bot devolverá una lista con las voces guardadas por el usuario. Clickando en una de esas voces, se activará como la voz utilizada para clonar.

Para eliminar una voz guardada, habrá que enviar el comando */deletevoice* y a continuación seleccionar una de las voces guardadas.

Bibliografía

[1] Anaconda. *Miniconda*. https://docs.anaconda.com/free/miniconda/. (2018). Último acceso: 06-04-2024.

[2] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin. *Attention is all you need*. https://arxiv.org/abs/1706.03762. (2017). Último acceso: 30-11-2023.

[3] James Betker. *Better speech synthesis through scaling*. (2022). Último acceso: 30-11-2023. 2023. arXiv: 2305.07243 [cs.SD].

[4] BotoStore. *BotFather*. https://botostore.com/c/botfather/. (2023). Último acceso: 14-12-2023.

[5] Eduardo Muñoz. *Attention is all you need: Discovering the Transformer paper*. https://towardsdatascience.com/attention-is-all-you-need-discovering-the-transformer-paper-73e5ff5e0634. (2023). Último acceso: 15-05-2024.

[6] Hugging Face. *OpenAI GPT2*. https://huggingface.co/docs/transformers/model_doc/gpt2. Último acceso: 30-11-2023.

[7] Indently. *How To Create A Telegram Bot In Python For Beginners (2023 Tutorial)*. https://www.youtube.com/watch?v=vZtm1wuA2yc. (2023). Último acceso: 16-04-2024.

[8] James Betker. *TorToiSe Architectural Design Doc*. https://nonint.com/2022/04/25/tortoise-architectural-design-doc/. (2022). Último acceso: 30-11-2023.

[9] James Betker. *TorToiSe text-to-speech*. https://github.com/neonbjb/tortoise-tts. (2022). Último acceso: 30-11-2023.

[10] Leandro Toledo. *python-telegram-bot*. https://docs.python-telegram-bot.org/en/v20.7/. (2023). Último acceso: 01-12-2023.

[11] Online Gantt. *Free Online Gantt*. https://www.onlinegantt.com/#/gantt. (2023). Último acceso: 10-12-2023.

[12] PMI. *A Guide to the Project Management Body of Knowledge (PMBOK Guide), 4th Edition*. Project Management Institute Inc., 2008.

[13] PyPI. *pydub*. https://pypi.org/project/pydub/. (2023). Último acceso: 18-12-2023.

[14] PyPI. *Speech Recognition*. https://pypi.org/project/SpeechRecognition/. (2023). Último acceso: 18-12-2023.

[15] PyPI. *torchaudio*. https://pypi.org/project/torchaudio/. (2023). Último acceso: 18-12-2023.

[16] *Python*. https://www.python.org/. (2024). Último acceso: 08-04-2024.

[17] Python. *math - Mathematical functions*. https://docs.python.org/3/library/math.html. (2023). Último acceso: 18-12-2023.

[18] Python. *os — Miscellaneous operating system interfaces*. https://docs.python.org/3/library/os.html. (2023). Último acceso: 18-12-2023.

[19] Python. *shutil — High-level file operations*. https://docs.python.org/3/library/shutil.html. (2023). Último acceso: 18-12-2023.

[20] Python. *typing — Support for type hints*. https://docs.python.org/3/library/typing.html. (2023). Último acceso: 18-12-2023.

[21] Bob Hughes Mike Cotterell Rajib Mall. *Software Project Management*. Mc Graw-Hill Education, 2009.

[22] Scrum. *What is Scrum?* https://www.scrum.org/learning-series/what-is-scrum. Último acceso: 30-11-2023.

[23] Telegram. *BotFather*. https://t.me/botfather. (2023). Último acceso: 14-12-2023.

[24] Wikipedia. *Archivo:Visual Studio Code 1.35 icon.svg*. https://es.m.wikipedia.org/wiki/Archivo:Visual_Studio_Code_1.35_icon.svg. Último acceso: 30-11-2023.

[25] Wikipedia. *Chain-of-responsibility pattern*. https://en.wikipedia.org/wiki/Chain-of-responsibility_pattern. (2023). Último acceso: 01-12-2023.

[26] Wikipedia. *Chatbot*. https://en.wikipedia.org/wiki/Chatbot. (2024). Último acceso: 08-04-2024.

[27] Wikipedia. *Diagrama de flujo*. https://es.wikipedia.org/wiki/Diagrama_de_flujo. (2024). Último acceso: 09-04-2024.

[28] Wikipedia. *Historia de Python*. https://es.wikipedia.org/wiki/Historia_de_Python. Último acceso: 30-11-2023.

[29] Wikipedia. *Lean software development*. https://es.wikipedia.org/wiki/Lean_software_development. (2023). Último acceso: 10-12-2023.

[30] Wikipedia. *Método de desarrollo de sistemas dinámicos*. https://es.wikipedia.org/wiki/M%C3%A9todo_de_desarrollo_de_sistemas_din%C3%A1micos. (2023). Último acceso: 10-12-2023.

[31] Wikipedia. *Programación extrema*. https://es.wikipedia.org/wiki/Programaci%C3%B3n_extrema. (2023). Último acceso: 10-12-2023.

[32] Wikipedia. *Telegram (software)*. https://en.wikipedia.org/wiki/Telegram_(software). (2024). Último acceso: 08-04-2024.

[33] Won Jang, Dan Lim, Jaesam Yoon, Bongwan Kim, Juntae Kim. *UnivNet: A Neural Vocoder with Multi-Resolution Spectrogram Discriminators for High-Fidelity Waveform Generation*. https://arxiv.org/abs/2106.07889. (2021). Último acceso: 30-11-2023.