



CENTRO DE INVESTIGACIÓN Y DE ESTUDIOS AVANZADOS
DEL INSTITUTO POLITÉCNICO NACIONAL

Unidad Zacatenco
Departamento de Computación

**Sistema de reconocimiento de voz y del habla
multi-fuente orientado a compañeros digitales**

Tesis que presenta
Ivan Giovanni Valdespin Garcia
para obtener el Grado de
Maestro en Ciencias
en Computación

Directores de Tesis
Dra. Sonia Guadalupe Mendoza Chapa
Dr. Luis Martín Sánchez Adame

Ciudad de México

Diciembre de 2023

Agradecimientos

Quiero agradecer a las siguientes personas e instituciones que me han apoyado a lo largo del camino para el cumplimiento de este proyecto de tesis:

- Estoy totalmente agradecido con el Consejo Nacional de Humanidades, Ciencias y Tecnologías (CONAHCyT) por haberme otorgado la beca que ayudó, a lo largo de estos dos años, a poder sustentar mis necesidades para lograr este proyecto satisfactoriamente.
- Al Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional (CINVESTAV) y al Departamento de Computación, por brindarme los conocimientos requeridos para la elaboración del proyecto de tesis, así como las aulas requeridas para el aprendizaje de los conocimientos necesarios.
- A la Universidad de St. Gallen en Suiza por abrirme sus puertas y permitirme llevar a cabo la culminación de mi proyecto de tesis, brindándome todos los recursos materiales como económicos para poder sustentar tanto mi proyecto como mi estancia en Suiza, muchas gracias.
- A mi asesora de tesis, la Dra. Sonia G. Mendoza Chapa por brindarme la oportunidad de culminar mi proyecto de tesis en el extranjero, así como por todo el apoyo durante la elaboración de mi proyecto de tesis, muchas gracias.
- A la Dra. Kimberly García de la Universidad de St. Gallen, Suiza, por todo el apoyo que me brindó durante mi estancia, muchas gracias.
- A todos los profesores por brindarme muchas enseñanzas tanto académicas como de la vida diaria. Asimismo al personal de apoyo que siempre estuvo en disposición para brindar su ayuda cuando fuera necesario.
- A mis padres y hermanos por siempre estar ahí en todo momento apoyándome en buenos y malos ratos y por animarme a seguir con mis sueños y no desistir hasta lograrlos.

- Finalmente, pero no menos importante, a mis amigos que han estado durante y hasta la culminación de este proyecto, viviendo muchas historias que permanecerán en mi mente, muchas gracias.

Resumen

Durante los últimos años, se han logrado grandes avances tecnológicos en múltiples sectores. Uno de ellos es el hogar, donde es cada vez más común encontrar diversos dispositivos inteligentes, como televisores y refrigeradores, y más recientemente asistentes personales controlados por voz. Estos últimos han ganado popularidad debido a su potencial para realizar múltiples tareas dentro del hogar, como controlar aparatos electrónicos, gestionar la iluminación y recordar eventos importantes. No obstante, su utilidad va más allá, ya que pueden ser utilizados en diferentes ámbitos, como en el sector industrial o incluso en el sector de la salud. Existen propuestas que, aprovechando los asistentes más populares en la actualidad, como Alexa y Google Assistant, ayudan a gestionar enfermedades mediante recordatorios de medicamentos o la programación de citas médicas, con el objetivo de mejorar la calidad de vida de los pacientes. Sin embargo, todos estos asistentes sufren de algunas deficiencias notables. La carencia de conciencia contextual es uno de los problemas, ya que estos asistentes requieren que el usuario esté a una distancia relativamente corta para reconocer los comandos, sin poder adaptarse a situaciones en las que el usuario esté en un lugar más alejado dentro del mismo entorno (e.g., en otra habitación del hogar). Esta limitación se debe a que estos asistentes utilizan una sola fuente de entrada. Otra de las deficiencias de estos asistentes es que la información del usuario es compartida con las grandes compañías que los han desarrollado; debido a que todo el procesamiento de la petición del usuario se realiza utilizando cómputo en la nube, la información proporcionada por el usuario tiende a ser enviada a la base de información de estas compañías como Google o Amazon, permitiendo que estas puedan tener acceso a la información personal de todos estos usuarios. En este trabajo de tesis, se propone, como solución a estas deficiencias, el desarrollo de un sistema de reconocimiento de voz y habla que admita múltiples fuentes de captación de audio (multi-fuente), lo que permitirá crear un entorno consciente del contexto. Asimismo, con el objetivo de mantener la información del usuario en un entorno local, el sistema de reconocimiento de voz y habla está basado en la utilización de herramientas que no requieran del cómputo en la nube, evitando que la información sea compartida con alguna compañía. Finalmente, este sistema es versátil, ya que puede interactuar con distintos sistemas externos (e.g., el *broker* de un compañero digital o dispositivos del Internet de las Cosas) haciéndolo capaz de ser implementado en distintos casos de estudio.

Palabras clave: asistentes personales comandados por voz, captura multi-fuente de audio, entorno consciente de contexto, reconocimiento de voz y habla.

Abstract

Over the last few years, great technological advances have been made in multiple sectors, specially in domestic scenarios, where it is increasingly common to find various smart devices, such as televisions and refrigerators, and more recently voice-controlled personal assistants. The latter have gained popularity due to their potential to accomplish multiple tasks within the home, such as controlling electronic appliances, managing lighting and remembering important events. However, their practicality goes beyond that, as they can be used in different fields, such as the industrial or healthcare sectors. There are proposals that, taking advantage of the most popular assistants today, such as Alexa and Google Assistant, help manage diseases through reminders about scheduling medical appointments or medication intake, to improve patients' quality of life. However, all of these assistants suffer from some notable shortcomings, notably, lack of contextual awareness. Typically, users must be within close proximity for these devices to recognize commands, limiting their functionality in larger spaces, such as different rooms within a house. This is a consequence of relying on a single audio input source. Additionally, privacy concerns arise as these devices, utilizing cloud computing, often share user data with their parent companies, such as Google or Amazon. In this thesis, we propose a solution to these deficiencies, through the development of a speech and voice recognition system that supports multiple audio sources, which will allow the creation of a context-aware environment. Moreover, in order to keep the user's information private, the speech and voice recognition system is confined to the local environment, operating independently of cloud computing. The system is also versatile, allowing interaction with different external systems, including a digital companion broker and Internet of Things devices, rendering it capable of being implemented in different case studies.

Keywords: context-aware environment, multi-source audio capture, speech and voice recognition, voice-controlled personal assistants.

Índice general

1. Introducción	1
1.1. Motivación	2
1.2. Planteamiento del problema	3
1.3. Hipótesis	4
1.4. Objetivos general y específicos del proyecto	5
1.4.1. Objetivo general	5
1.4.2. Objetivos específicos	5
1.5. Organización del documento	5
2. Estado del arte	7
2.1. Marco teórico	7
2.1.1. Filtros digitales	7
2.1.2. Transformada Rápida de Fourier	11
2.1.3. Redes neuronales convolucionales (CNN)	12
2.1.4. Matriz de confusión	13
2.1.5. Protocolo de comunicación HTTP	15
2.2. Trabajos relacionados	18
3. Análisis y diseño del sistema de reconocimiento de voz y habla multi-fuente	29
3.1. Aplicación Android	29
3.1.1. Funcionamiento de la aplicación	29
3.1.2. Requerimientos	30
3.1.3. Casos de uso	32
3.1.4. Clases de la aplicación Android	37
3.2. Sistema de captura permanente	40
3.2.1. Funcionamiento del sistema	40
3.2.2. Requerimientos	41
3.2.3. Casos de uso	43
3.2.4. Diseño de solución	47
3.3. Micro-servicio para el reconocimiento de voz y habla	50

3.3.1.	Descripción del micro-servicio	51
3.3.2.	Módulo de identificación de usuarios	53
3.3.3.	Módulo de procesamiento de lenguaje natural	60
3.3.4.	Módulo de comunicación	65
3.3.5.	Clases del micro-servicio	69
4.	Implementación del sistema de reconocimiento de voz y habla multi-	73
	fuentes	
4.1.	Aplicación Android	73
4.1.1.	Interfaz de usuario	73
4.1.2.	Aspectos técnicos de la aplicación	78
4.2.	Sistema de captura permanente	85
4.2.1.	Hardware seleccionado	86
4.2.2.	Frameworks y bibliotecas utilizadas	90
4.2.3.	Aspectos técnicos del sistema	91
4.2.4.	Palabra de activación	99
4.2.5.	Rango de captura de audio del sistema de captura permanente . . .	101
4.3.	Micro-Servicio	102
4.3.1.	Selección de dispositivo para el micro-servicio	103
4.3.2.	Frameworks	105
4.3.3.	Inicialización del micro-servicio	113
4.3.4.	Generación de repositorios	117
4.3.5.	Manipulación y procesamiento de archivos de audio	119
4.3.6.	Generación del modelo de identificación de usuarios	122
4.3.7.	Identificación de petición y extracción de información relevante . . .	129
4.3.8.	Envío de comandos	131
5.	Casos de estudio	135
5.1.	Compañero digital para el cuidado de la diabetes	135
5.1.1.	Descripción general	135
5.1.2.	Identificación de peticiones y extracción de información para la co- municación con el <i>Broker Service</i>	139
5.2.	Sistema para la interacción con dispositivos de Internet de las Cosas	144
5.2.1.	Descripción general	144
5.2.2.	Identificación de peticiones y extracción de información para la co- municación con dispositivos del Internet de las Cosas	145
5.3.	Sistema de identificación de objetos basada en contexto	149
5.3.1.	Descripción general	150
5.3.2.	Integración del sistema de reconocimiento de voz y habla multi- fuente con el sistema OIC	151

6. Pruebas y resultados	157
6.1. Compañero digital para el cuidado de la diabetes	157
6.1.1. Identificación de peticiones	157
6.1.2. Obtención de características	160
6.1.3. Tiempo de respuesta del sistema	161
6.2. Interacción con dispositivos del Internet de las Cosas	162
6.2.1. Identificación de dispositivos del Internet de las Cosas	162
6.2.2. Tiempo de respuesta del sistema	163
6.3. Integración con el sistema OIC	165
7. Conclusiones y trabajo a futuro	171
7.1. Conclusiones	172
7.2. Trabajo futuro	173

Índice de figuras

1.1. Diagrama de la organización del documento de tesis.	6
2.1. Aplicación de la función <i>Max-Pooling</i> a un vector de dimensionalidad seis .	13
2.2. Aplicación de la función <i>Average Pooling</i> a un vector de dimensionalidad seis	13
2.3. Matriz de confusión	14
2.4. Arquitectura del sistema mostrando la organización direccional de entradas y salidas de datos [Cheng et al., 2018]	24
3.1. Diagrama de casos de uso de la aplicación Android.	32
3.2. Diagrama de clases de aplicación Android	38
3.3. Diagrama de casos de uso del sistema de captura permanente	44
3.4. Diagrama general de la arquitectura del sistema de captura permanente . .	49
3.5. Diagrama general de la arquitectura del micro-servicio	52
3.6. Diagrama de casos de uso del módulo de identificación de usuarios	56
3.7. Diagrama de casos de uso del módulo de procesamiento de lenguaje natural	62
3.8. Diagrama de casos de uso del módulo de comunicación	67
3.9. Diagrama de clases del micro-servicio para el reconocimiento de voz y habla multi-fuente orientado a compañeros digitales.	69
4.1. Componentes de la primera versión de la aplicación para la captura y envío de peticiones.	74
4.2. Componentes de la versión final de la aplicación para la captura y envío de peticiones.	76
4.3. Mensajes de interacción para guiar al usuario (1/3).	77
4.4. Mensajes de interacción para guiar al usuario (2/3).	77
4.5. Mensajes de interacción para guiar al usuario (3/3).	78
4.6. Constructor de la clase <code>AudioMP3</code> donde se definen los atributos de la clase	79
4.7. Características utilizadas en el método <code>prepareRecord</code> para la configura- ción de una instancia de <code>MediaRecorder</code>	80
4.8. Implementación de la clase <code>CreateAudioFileFactory</code> para la generación de archivos de audio	81

4.9. Mensaje de error de conexión con el micro-servicio desplegado en el cuadro de texto de la aplicación.	82
4.10. Implementación del método <code>onResponse</code> para recibir las respuestas del micro-servicio	82
4.11. Implementación de los métodos <code>makeBigger</code> y <code>makeSmaller</code> para la manipulación de la fuente de los mensajes de texto	83
4.12. Implementación del método <code>onInit</code> para la inicialización de una instancia de la clase <code>Reader</code>	84
4.13. Implementación del método <code>getInstance</code> para la generación del patrón de diseño <i>Singleton</i> de la clase <code>Logger</code>	85
4.14. Señales de audio a 40dB y 50dB utilizando el micrófono Adafruit AGC-MAX9814	88
4.15. Señal de audio a 60dB capturada por el micrófono Adafruit AGC-MAX9814	88
4.16. Diseño de filtro analógico pasa bajas para eliminación de ruido	89
4.17. <i>Hardware</i> utilizado en la implementación del sistema de captura permanente	90
4.18. Archivo de configuración del sistema de captura permanente	91
4.19. Parámetros de configuración del protocolo I2S para la inicialización del micrófono digital.	93
4.20. Generación de las tareas del sistema de captura permanente	95
4.21. Implementación de la tarea <code>WakeWordTask</code>	96
4.22. Encendido del LED indicador para comenzar a grabar la petición	97
4.23. Tarea <code>LigtherTask</code> encargada del manejo del LED indicador	97
4.24. Captura de petición de usuario mediante sistema de captura permanente .	99
4.25. Comunicación entre el sistema de captura permanente y el micro-servicio .	99
4.27. Porcentaje de precisión en la detección de la palabra de activación a diferentes distancias.	102
4.28. Resultados obtenidos en pruebas de traducción, utilizando los diferentes modelos de <code>Whisper</code>	106
4.29. Resultados obtenidos en pruebas de tiempo de traducción del <i>framework</i> <code>Whisper</code> utilizando una <i>laptop</i> y cuatro hilos	107
4.30. Resultados obtenidos en pruebas de tiempo de traducción del <i>framework</i> <code>Whisper</code> utilizando la tarjeta <i>RaspBerry Pi 4B</i> y cuatro hilos	107
4.31. Comparativa de tiempos de traducción de archivos de audio utilizando <code>Whisper</code> con una implementación de cuatro hilos y con <code>OpenVINO</code>	109
4.32. Comparativa de pruebas de velocidad de modelos de <code>spaCy</code> en dos dispositivos diferentes	111
4.33. Archivo de configuración <code>MainConfig.json</code>	113
4.34. Información contenida en el archivo <code>EnglishConfig.json</code>	114
4.35. Método de la clase <code>MainConfiguration</code> para inicializar <code>Whisper</code>	115
4.36. Método de la clase <code>MainConfiguration</code> para inicializar <code>SpaCy</code>	116
4.37. Interfaz utilizada para la generación de la clases <code>EnglishPattern</code> y <code>SpanishPattern</code>	116

4.38. Método para inicialización de los módulos del micro-servicio	117
4.39. Inicialización del micro-servicio en idioma español	117
4.40. Implementación del patrón de diseño Factory para la generación de repositorios	118
4.41. Generación de repositorios de usuarios para almacenar muestras de audio .	119
4.42. Método de la clase AudioFile para la segmentación de archivos de audio .	120
4.43. Clase implementada para la generación de filtro paso banda	121
4.44. Constructor de la clase Model para la generación del modelo de identificación de usuarios	123
4.45. Arquitectura del bloque residual utilizado en la arquitectura RestNet de la red convolucional	125
4.46. Arquitectura RestNet de la red convolucional para la generación del modelo de identificación de personas	126
4.47. Ejemplo de la generación de un patrón para la identificación de elementos, utilizando el <i>framework</i> SpaCy	130
4.48. Implementación del patrón de diseño Factory para la generación de instancias de dispositivos	131
4.49. Implementación de la clase BrokerCommunication	132
4.50. Prueba de comunicación del micro-servicio con un sistema <i>broker</i> utilizando el framework Kafka	132
5.1. Diagrama general de la arquitectura de compañeros digitales para la atención a pacientes con enfermedades crónicas.	137
5.2. Oraciones utilizadas para identificación del tipo de peticiones	140
5.3. Patrones diseñados para la extracción de información relevante en español .	141
5.4. Prueba de extracción de características de recordatorio de medicamento . .	142
5.5. Formato <i>json</i> para la interacción con el <i>Broker Service</i>	143
5.6. Inicialización y recibimiento de información por parte del <i>broker</i> implementado de manera local, utilizando el <i>framework Kafka</i>	144
5.7. Patrones generados para la identificación del dispositivo a utilizar	146
5.8. Lista de palabras para la identificación de dispositivos y extracción de información relevante.	147
5.9. Patrones diseñados para la extracción de información relevante	148
5.10. Mensajes preestablecidos para la interacción con el usuario	149
5.11. <i>Headers</i> establecidos para el envío de <i>buffers</i> de audio	152
5.12. <i>Endpoint</i> “Audio” para la captura y procesamiento de <i>buffers</i> de audio relacionados con el sistema OIC	153
5.13. Generación de directorios y almacenamiento de archivos recibidos para la integración del micro-servicio con el sistema OIC.	153
5.14. Generación del archivo de texto referente a la traducción del audio obtenido por sistemas de captura permanente	154

- 6.1. Comparativa del tiempo de interacción con dispositivos de Internet de las Cosas, utilizando diferentes fuentes de entrada 164
- 6.2. Ejemplificación del escenario realizado para pruebas de integración del sistema de reconocimiento de voz y habla multi-fuente con el sistema OIC . . 166

Índice de tablas

2.1. Componentes detallados de un filtro y sus funciones.	9
2.2. Evaluación del sistema reportado por [Rehman et al., 2020]	21
2.3. Resultados obtenidos por [Alagha and Helbing, 2019] sobre la evaluación de asistentes digitales	25
2.4. Comparativa de los trabajos reportados en el estado del arte	26
4.1. Comparativa de dispositivos de bajo consumo para selección de dispositivos de entrada del sistema	86
4.2. Comparativa entre los micrófonos Adafruit AGC-MAX9814 y INMP441 . . .	87
4.3. Comparativa de evaluación entre palabras de activación	100
4.4. Pruebas de identificación de la palabra de activación a distintas distancias . . .	101
4.5. Comparativa de características de VIPAs actuales en el mercado	103
4.6. Comparativa de características de tarjetas de desarrollo consideradas para implementación del reconocedor de voz y habla	104
4.7. Comparación entre Apache Kafka y Apache ActiveMQ	112
4.8. Resultados de las pruebas de velocidad y exactitud del entrenamiento del modelo de identificación de personas	126
4.9. Matriz de confusión para la identificación de dos usuarios	127
4.10. Matriz de confusión para la identificación de tres usuarios	128
6.1. Primera matriz de confusión para la evaluación de la identificación del tipo de peticiones.	158
6.2. Primeros resultados de evaluación del sistema de reconocimiento de voz y habla para la identificación de peticiones	158
6.3. Segunda matriz de confusión para la evaluación de la identificación del tipo de peticiones.	159
6.4. Segundos resultados de evaluación del sistema de reconocimiento de voz y habla para la identificación de peticiones	159
6.5. Matriz de confusión de la característica “Médico”	160
6.6. Matriz de confusión de la característica “Fecha”	160
6.7. Matriz de confusión de la característica “Tiempo”	160
6.8. Matriz de confusión de la característica “Buscar información”	160

6.9. Resultados de evaluación del sistema de reconocimiento de voz y habla para la identificación de características	160
6.10. Comparativa del tiempo de respuesta entre la aplicación Android y el sistema de captura permanente	161
6.11. Matriz de confusión del dispositivo “Robot”	162
6.12. Matriz de confusión del dispositivo “Cortinas”	162
6.13. Matriz de confusión del dispositivo “Lámpara”	162
6.14. Resultados de la evaluación del sistema de reconocimiento de voz y habla para la identificación de características	163
6.15. Comparativa del tiempo de respuesta utilizando aplicación Android	164
6.16. Comparativa del tiempo de respuesta utilizando ESP32-S3	164

Capítulo 1

Introducción

Actualmente, existen sistemas que asisten a las personas en diversas tareas de diferentes ámbitos, interactuando con ellos simplemente a través de comandos de voz. Estos sistemas, también conocidos como compañeros digitales, pueden clasificarse en tres tipos de roles: guardianes, asistentes o mentores y compañeros [Kritzler et al., 2019]. Su uso se observa comúnmente en hogares con asistentes como *Alexa* o *Google Assistant*, que pueden realizar tareas como encender luces, interactuar con dispositivos electrónicos (e.g., televisión y lavadora), buscar información e incluso configurar recordatorios de citas. Debido a su potencial de implementación en varios sectores, se han desarrollado propuestas de compañeros digitales en la industria [Kritzler et al., 2019] y en el ámbito médico [Karimi et al., 2022] [Dojchinovski et al., 2019], entre otros.

A pesar de los beneficios que ofrecen estos sistemas, como información y servicios [Garcia et al., 2020] que mejoran la vida diaria de los usuarios, los compañeros digitales actuales y los asistentes propuestos en la literatura científica, como los reportados en [Ermolina, 2021], [Karimi et al., 2022] y [Dojchinovski et al., 2019], sufren de ciertas deficiencias. Una de estas limitaciones es la falta de conciencia del contexto, término que se define como "la capacidad que tiene un sistema para poder obtener, interpretar y utilizar información acerca de su entorno, y adaptar su funcionalidad dependiendo del contexto actual, con el fin de proporcionar los servicios apropiados al usuario" [Hong et al., 2009]. Se considera como un factor clave en el desarrollo y difusión de ambientes inteligentes [Avola, 2009], pues implica la identificación de dispositivos y anfitriones cercanos, así como los cambios en estos elementos a lo largo del tiempo [Mensah and Mwakapesa, 2022]. Todos los asistentes comerciales y los nuevos compañeros digitales propuestos en la academia utilizan un sistema mono-fuente, i.e., el sistema sólo captura audio del mismo asistente. En consecuencia, los usuarios deben estar cerca del asistente para dar instrucciones. Dependiendo de la posición del usuario, dar instrucciones claras al asistente puede ser desafiante o, a veces, la instrucción puede no entenderse claramente. Con base en esta limitación y en la definición de conciencia del contexto mencionada anteriormente, se

puede decir que estos sistemas carecen de ella.

En el contexto de los sistemas de compañeros digitales, la privacidad, especialmente en términos de confidencialidad de datos, resalta como una preocupación crítica. La confidencialidad, uno de los principios clásicos de la seguridad de la información, se refiere a la protección de datos personales [Whitman and Mattord, 2021], garantizando que sólo sean accesibles para entidades autorizadas [National Institute of Standards and Technology, 2004]. Este principio es crucial en sistemas donde los usuarios comparten información sensible, como en los asistentes virtuales y compañeros digitales. La información suministrada por los usuarios a estos sistemas, necesaria para el procesamiento en tiempo real de solicitudes, a menudo se comparte con las empresas desarrolladoras, como Amazon y Google. Esta transmisión de datos implica el uso de servicios de computación en la nube, ejemplificado por el uso de *AWS* por parte de *Alexa* y *Google Cloud* por *Google Assistant*. Consecuentemente, detalles personales como los historiales de búsqueda en Internet y patrones de interacción social son compartidos con estas compañías [Amazon, 2019], [goo, 2023]. De esta manera, es esencial que los usuarios tengan el derecho de no compartir sus datos, incluyendo sus metadatos, con ninguna entidad [Munier et al., 2014]. Este derecho a la privacidad y al control absoluto sobre los datos personales es un pilar de la ética digital y la gobernanza de datos [Floridi, 2018]. La salvaguarda de este derecho es clave para preservar la integridad y la confianza en el uso de tecnologías digitales [Palermos, 2023].

1.1. Motivación

Los asistentes personales comandados por voz (VIPAs por sus siglas en inglés) han tomado fuerza con el paso del tiempo, ya que el grado de aceptación por parte de los usuarios, incluyendo a los adultos mayores, ha ido en aumento y se espera que, en años futuros, la cantidad de personas que utilicen estos dispositivos sea mayor [Dojchinovski et al., 2019], [Alagha and Helbing, 2019]. Esta tendencia se debe, en gran parte, a la forma relativamente intuitiva de interacción entre el usuario y el dispositivo, haciendo que sea más sencillo para cualquier persona poder utilizarlos y habituarse a ellos [Brewer et al., 2022].

Actualmente, los VIPAs se encuentran inmersos en nuestra vida cotidiana, mediante dispositivos como *smartphones*, televisores inteligentes y electrodomésticos. Incluso grandes compañías ofrecen este tipo de asistentes, tales como *Google Assistant* de Google, *Alexa* de Amazon y *Siri* de Apple [Salai et al., 2021], que nos permiten, con sólo dar una instrucción, realizar una gran variedad de tareas como buscar información en Internet y programar alarmas, y más recientemente sirven como apoyo en el cuidado de la salud [Ermolina, 2021].

Aunque los VIPAs enfocados en el cuidado de la salud aún se encuentran en sus inicios,

en comparación con los VIPAs centrados en el entretenimiento, [Ermolina, 2021], existen múltiples propuestas que ayudan a llevar un mejor control sobre los cuidados que necesita una persona con alguna enfermedad. Sin embargo, la mayoría de estas propuestas son dependientes de asistentes comerciales (como los ya mencionados *Alexa*, *Google Assistant* y *Siri*) para el reconocimiento y procesamiento de los comandos de voz. Un ejemplo de ello es el trabajo reportado en [Dojchinovski et al., 2019] en donde proponen un VIPA utilizando los servicios de *Alexa* y *Google Assistant* para el constante monitoreo de los signos vitales de una persona en el hogar.

Este proyecto propone un sistema de reconocimiento de voz y habla, que aborda los dos problemas mencionados anteriormente. El sistema multi-fuente desarrollado es capaz de atender las solicitudes de los usuarios sin requerir que estén cerca del dispositivo central, generando así una conciencia del contexto. Además, mantiene la información del usuario de manera privada, ya que no depende de herramientas de computación en la nube y, por lo tanto, no comparte información con empresas externas o individuos. Esto se logra utilizando herramientas como *Whisper* [OpenAI, 2022] y *spaCy* [Honnibal et al., 2020a], que se pueden utilizar localmente.

1.2. Planteamiento del problema

Los asistentes propuestos en la literatura científica utilizan, como elemento de adquisición de audio, los mismos asistentes que ya se encuentran en el mercado, tales como *Alexa*, *Google Assistant* y *Siri*, teniendo así un sistema mono-fuente (i.e., el sistema sólo tiene como captura de audio al mismo asistente). En consecuencia, se genera la necesidad de localizarse cerca del asistente para poder darle instrucciones, ya que, de otro modo, puede resultar complicado dar una instrucción al mismo. Aunado a esto, estudios como los de [Alagha and Helbing, 2019] y [Brewer et al., 2022] mencionan la dificultad que puede llegar a tener este tipo de asistentes para la comprensión de instrucciones relacionadas con el cuidado de la salud. Por ejemplo, el correcto funcionamiento del asistente se ve afectado cuando la persona no se encuentra a una distancia, lo suficientemente cercana del asistente, para que éste logre capturar la instrucción de manera clara.

Las complicaciones expuestas sugieren que existe una falta de conciencia del contexto, ya que no cuentan con la capacidad de entender, interpretar o utilizar información acerca del entorno y adaptar su funcionalidad consecuentemente para proporcionar al usuario los servicios apropiados [Hong et al., 2009]. Estos asistentes no son capaces de obtener información del entorno (e.g., el hogar) en donde se pretende que el dispositivo funcione adecuadamente y se limitan a un funcionamiento dentro de un rango de distancia delimitado por la capacidad de adquisición de audio de los propios asistentes. Esto hace que el

usuario se vea obligado a estar siempre dentro de un área cercana al dispositivo e impide poder interactuar con él, si el usuario se encuentra en una sección del hogar en donde no se encuentre el asistente.

En este proyecto se realizó un sistema de reconocimiento de voz y habla multi-fuente (i.e., el sistema tiene como captura de audio más de una fuente de entrada), el cual debe ser capaz de identificar a la persona, mediante el uso de herramientas de reconocimiento de voz. Una vez que el reconocedor ha detectado a la persona, éste identifica comandos de voz incluidos en el lenguaje natural que el usuario le proporcione.

Además, el sistema tiene la capacidad de utilizar varias fuentes de audio para identificar la tarea a realizar. Estas fuentes de audio envía la señal adquirida a un dispositivo basado en Raspberry (Rpi) o alguna plataforma basada en microcontroladores.

Las fuentes de captura del audio pueden ser smartphones (mediante un servicio, o una *app* activa en el mismo), pulseras inteligentes con micrófono (mediante una señal compartida por el *wearable* como bluetooth o wi-fi) y el mismo dispositivo basado en Rpi.

Las múltiples fuentes de captura de audio y el sistema de reconocimiento de voz permiten generar un ambiente consciente de contexto, ya que el sistema es consciente de la presencia del usuario y su ubicación.

El sistema es capaz de utilizar bibliotecas o proyectos como Julius [Julius, 2014], que ayuden a la parte del procesamiento del lenguaje natural. Por lo tanto, la parte de responsabilidad del sistema es el procesamiento adecuado de los comandos detectados en el habla del usuario y la generación de un envoltorio para que dichos comandos sean transmitidos a un sistema encolador de peticiones.

1.3. Hipótesis

Es posible desarrollar un reconocedor de voz multi-fuente capaz de interactuar con los usuarios en diversos dominios de aplicación, asegurando la confidencialidad mediante el procesamiento integral de datos de forma local, sin la necesidad de compartir información con entidades externas.

1.4. Objetivos general y específicos del proyecto

1.4.1. Objetivo general

- Diseñar e implementar un reconocedor de voz y habla multi-fuente capaz de interactuar con un usuario en tres contextos específicos: (1) un compañero digital para el cuidado de la diabetes, (2) un sistema para la interacción con dispositivos del Internet de las Cosas y (3) un sistema de identificación de objetos basado en contexto. Este reconocedor de voz se desarrollará garantizando la confidencialidad de los datos del usuario mediante el procesamiento local, evitando la necesidad de compartir información con terceros.

1.4.2. Objetivos específicos

1. Determinar el dispositivo central apropiado para la gestión de las múltiples fuentes de entrada y la implementación de herramientas de procesamiento de lenguaje natural.
2. Definir las fuentes de entrada para el procesamiento de los comandos por el dispositivo central.
3. Definir un comando de activación para algunas fuentes de entrada que permitan la captura de la petición de un usuario.
4. Definir casos de estudio que permitan mostrar la versatilidad del reconocedor de voz y habla.
5. Realizar pruebas de precisión y exactitud del reconocedor de voz y habla para la identificación de los comandos establecidos en cada caso de estudio, mediante matrices de confusión.

1.5. Organización del documento

Esta tesis está organizada de la siguiente manera. En el Capítulo 2 se enfoca en el estado del arte, el cual se ha estructurado en dos partes principales: marco teórico y análisis de trabajos relacionados. En el Capítulo 3 se describe el análisis y diseño del sistema de reconocimiento de voz y habla, seguido del Capítulo 4 donde se explica la implementación de dicho sistema. El Capítulo 5 describe los casos de estudio desarrollados a partir del sistema propuesto. En el Capítulo 6 se tratan las pruebas y los resultados obtenidos. Finalmente, en el Capítulo 7, se abordan las conclusiones y el trabajo futuro.

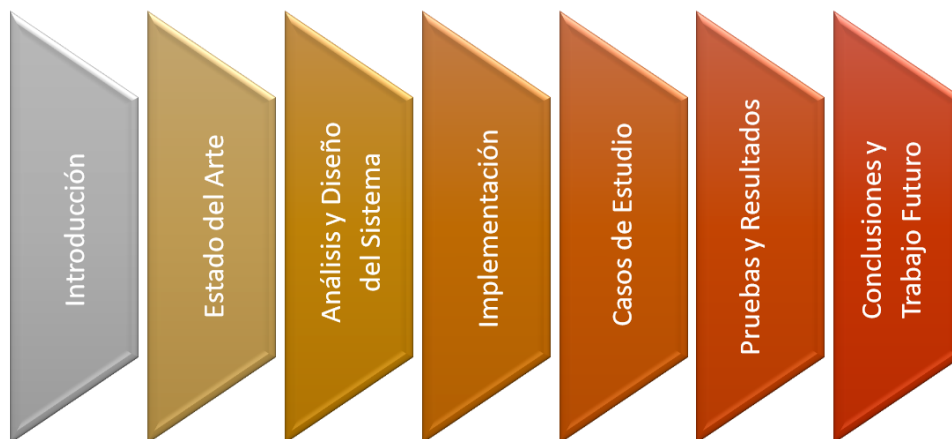


Figura 1.1: Diagrama de la organización del documento de tesis.

Capítulo 2

Estado del arte

En este capítulo se abordan los temas en los que se requiere tener conocimiento para la elaboración de este proyecto de investigación. En la Sección 2.1 se expone el marco teórico necesario para el desarrollo del sistema de reconocimiento de voz y habla multi-fuente orientado a compañeros digitales. A continuación, en la Sección 2.2, se analizan los principales trabajos relacionados.

2.1. Marco teórico

En este apartado se describen, de manera concisa, los elementos utilizados durante el desarrollo del sistema de reconocimiento de voz y habla multi-fuente orientado a compañeros digitales. En la Sección 2.1.1 se abordan los filtros digitales y en la Sección 2.1.2 se explica la Transformada Rápida de *Fourier*. Posteriormente, en la Sección 2.1.3 se describen las redes convolucionales y en la Sección 2.1.4 se aborda la Matriz de Confusión, junto con sus métricas de desempeño. Finalmente, en la Sección 2.1.5 se describe el protocolo de comunicación HTTP.

2.1.1. Filtros digitales

Un filtro digital constituye una herramienta matemática diseñada con el propósito de eliminar tipos específicos de ruido o artefactos presentes en señales digitales. Su funcionamiento radica en permitir el paso únicamente de determinadas frecuencias a través del filtro, al mismo tiempo que atenúa o suprime otras.

Estos filtros digitales se despliegan en una amplia gama de aplicaciones, abarcando desde el procesamiento de audio y de imágenes hasta las comunicaciones digitales. En cada contexto, el objetivo primordial es la eliminación de información no deseada de la señal, salvaguardando con precisión la información que se busca conservar.

Existen varios tipos de filtros digitales, cada uno con sus particularidades y ventajas. Entre los más comunes se encuentran los filtros de paso bajo, de paso alto y de paso banda. Los filtros de paso bajo se diseñan con el propósito de eliminar el ruido de alta frecuencia presente en una señal. Este tipo de filtros encuentra aplicaciones frecuentes en el ámbito del procesamiento de audio, donde se utilizan para reducir silbidos y otros artefactos de alta frecuencia. Contrariamente, los filtros de paso alto buscan eliminar el ruido de baja frecuencia en una señal. En el contexto del procesamiento de audio, se emplean para reducir el ruido de fondo y otros artefactos de baja frecuencia. Por último, los filtros de paso banda están concebidos para permitir únicamente el paso de un rango específico de frecuencias a través del filtro. En aplicaciones de audio, se emplean para aislar un instrumento particular o un rango vocal.

Estos filtros digitales constituyen una pieza fundamental en numerosas aplicaciones de procesamiento de señales digitales. Al diseñar con meticulosidad el filtro, se logra eliminar la información no deseada de una señal, preservando de manera efectiva la información deseada. [Proakis, 2006]

La intersección entre filtros e Inteligencia Artificial (IA) ha generado avances significativos en diversas áreas. Por ejemplo, en el procesamiento de imágenes, los filtros paso bajo son esenciales para eliminar el ruido y resaltar características importantes. En el ámbito de la visión por computadora, la aplicación de filtros puede mejorar la detección de objetos y la segmentación de imágenes.

En la IA de procesamiento de señales, los filtros son cruciales para la eliminación de ruido en datos, contribuyendo así a la mejora del rendimiento de algoritmos de aprendizaje automático. Además, los filtros paso banda pueden ser útiles para extraer características específicas de las señales, lo que resulta valioso en tareas como el reconocimiento de patrones y la clasificación de datos.

En el campo del procesamiento del lenguaje natural, los filtros también desempeñan un papel clave. Pueden ser utilizados para preprocesar datos de texto, eliminando palabras irrelevantes o ruido lingüístico, lo que facilita la tarea de los modelos de lenguaje.

La relación entre filtros y IA no sólo mejora la eficiencia de los algoritmos, sino que también abre la puerta a la creación de sistemas más adaptables y robustos. La capacidad de ajustar automáticamente los parámetros de los filtros, según las condiciones cambiantes del entorno, es un área emocionante de investigación.

Tabla 2.1: Componentes detallados de un filtro y sus funciones.

Componente	Descripción y Función
Entrada	Señal original que se desea filtrar.
Salida	Señal procesada después de aplicar el filtro.
Función de transferencia	Representación matemática que describe cómo el filtro afecta diferentes frecuencias.
Frecuencia de corte	Frecuencia donde el filtro comienza a afectar la amplitud de la señal.
Orden del filtro	Indica la complejidad y capacidad del filtro para atenuar frecuencias no deseadas.
Respuesta en frecuencia	Representación gráfica del impacto del filtro en diferentes frecuencias.

2.1.1.1. Filtros paso bajo

Con base en el trabajo de [Proakis, 2006], [Orfanidis, 2010] y [Smith, 1999], se obtienen las siguientes características de los filtros paso bajo:

- **Funcionamiento:** permiten el paso de frecuencias por debajo de una frecuencia de corte y atenúan las frecuencias por encima de ella.
- **Respuesta en frecuencia:** la función de transferencia de un filtro paso bajo se define como:

$$H(f) = \frac{1}{1 + \left(\frac{f}{f_c}\right)^{2n}} \quad (2.1)$$

donde f_c es la frecuencia de corte y n es el orden del filtro.

- **Aplicaciones:** reducción de ruido y suavizado de señales.
- **Implementación:** Eliminación de componentes de altas frecuencias.

2.1.1.2. Filtros paso alto

Con base en los trabajos de [Proakis, 2006], [Orfanidis, 2010] y [Smith, 1999], se obtienen las siguientes características de los filtros paso alto:

- **Funcionamiento:** permiten el paso de frecuencias por encima de una frecuencia de corte y atenúan las frecuencias por debajo de ella.
- **Respuesta en frecuencia:** la función de transferencia de un filtro paso alto se define de manera similar al paso bajo, invirtiendo la respuesta en frecuencia.

- **Aplicaciones:** detección de bordes y eliminación de componentes de baja frecuencia.
- **Implementación:** similar a filtros paso bajo.

2.1.1.3. Filtros paso banda

Con base en los trabajos de [Proakis, 2006], [Orfanidis, 2010] y [Smith, 1999], se obtienen las siguientes características de los filtros paso banda:

- **Funcionamiento:** permiten el paso de un rango específico de frecuencias y atenúan las frecuencias fuera de ese rango.
- **Respuesta en frecuencia:** la función de transferencia de un filtro paso banda es la combinación de las funciones de transferencia de un paso bajo y un paso alto.

$$H(f) = \frac{1}{1 + \left(\frac{f}{f_{c, \text{ bajo}}}\right)^{2n}} \cdot \frac{1}{1 + \left(\frac{f_{c, \text{ alto}}}{f}\right)^{2n}} \quad (2.2)$$

donde $f_{c, \text{ bajo}}$ y $f_{c, \text{ alto}}$ son las frecuencias de corte.

- **Aplicaciones:** comunicaciones y procesamiento de audio.
- **Implementación:** combinación de filtros paso bajo y paso alto.

2.1.1.4. Frecuencia de Nyquist

Con base en los trabajos de [Proakis, 2006], [Orfanidis, 2010] y [Smith, 1999], se obtienen las siguientes características de la frecuencia de *Nyquist*:

- **Concepto:** la frecuencia de Nyquist establece el límite superior para las frecuencias que pueden ser representadas, de manera fiable, en una señal digital.
- **Importancia:** evita el *aliasing* durante el muestreo digital.
- **Fórmula:**

$$f_{\text{Nyquist}} = \frac{f_{\text{muestreo}}}{2} \quad (2.3)$$

- **Aplicación:** determina la frecuencia máxima representable sin distorsiones.

En el contexto de la IA, los algoritmos adaptativos están ganando terreno. Estos algoritmos permiten que los filtros se ajusten en tiempo real, según las variaciones en los datos de entrada. Por ejemplo, en el procesamiento de señales, un filtro adaptativo podría modificar sus parámetros para adaptarse a cambios en la frecuencia o amplitud de la

señal, garantizando así un rendimiento constante incluso en condiciones variables.

La implementación de técnicas de aprendizaje automático, como redes neuronales, también ha llevado a avances significativos en la adaptabilidad de los filtros. Estos modelos pueden aprender patrones complejos en los datos y ajustar automáticamente los filtros según las características identificadas.

2.1.2. Transformada Rápida de Fourier

Para poder abordar la explicación del algoritmo de la Transformada Rápida de Fourier (FFT, por sus siglas en inglés - *Fast Fourier Transform*) es necesario comprender qué es la Transformada Discreta de Fourier (DFT, por sus siglas en inglés - *Discrete Fourier Transform*) ya que el objetivo de este algoritmo es disminuir la complejidad computacional de cálculo que requiere la DFT.

La DFT es un caso particular de la transformada de Fourier que ayuda a analizar y manipular señales digitales, en el dominio de la frecuencia, y está basada en la idea de la descomposición de la señal discreta en sus componentes frecuenciales. Evalúa frecuencias concretas, compuestas de N puntos separados por un intervalo de tiempo T calculando así la amplitud y la fase de cada componente de frecuencia que se encuentre presente en la señal [Seco, 2019]. La DFT es descrita por [Gan, 2020] como:

$$X(k) = \sum_{n=0}^{N-1} x(n) \exp(-i2\pi kn/N), k = 0, 1, 2, 3, \dots, N - 1 \quad (2.4)$$

Una vez descrita la DFT, se puede decir que la FFT es un algoritmo que acelera el cálculo de la transformada discreta de Fourier, atribuido al trabajo de *cooley1965algorithm*. Este algoritmo, utilizado hasta la actualidad, simplifica la complejidad de la DFT (cuya representación se muestra en la ecuación 2.4) ayudando a que el costo computacional no sea demasiado alto y así evitar que el procesamiento sea lento y poco práctico [Gan, 2020].

La complejidad computacional de la DFT es de $O(N^2)$ debido a que existen N salidas x_k , donde cada salida requiere la suma de N términos. Sin embargo, al utilizar la FFT, la complejidad del método disminuye a $O(N \log N)$, siendo así menos costosa computacionalmente. En la actualidad, existen varios algoritmos para el cálculo de la FFT, los cuales están basados en la descomposición del cálculo de la DFT de una secuencia de largo N .

[De la Fraga, 2001] realiza una clasificación de los tipos de algoritmos que calculan la FFT, los cuales son:

- Algoritmos de decimación de tiempo, en donde el tiempo es descompuesto en subsecuencias más pequeñas.

- Algoritmos de decimación en frecuencia, en donde se utilizan los coeficientes de la DFT para descomponerlos en subsecuencias más pequeñas.

2.1.3. Redes neuronales convolucionales (CNN)

Las redes neuronales convolucionales son redes que están diseñadas para el procesamiento de datos que tienen una topología de cuadrilla ya que, de esta manera, las redes logran reducir la cantidad de parámetros a procesar, a diferencia de las redes neuronales totalmente conectadas [Habibi Aghdam and Jahani Heravi, 2017].

Generalmente, este tipo de redes son utilizadas para el procesamiento de imágenes, ya que se adecua de buena manera debido a su forma de dos dimensiones; sin embargo, estas redes pueden ser utilizadas también para el análisis de señales de una dimensión como lo son las señales de audio, señales cardiacas, etc. [Goodfellow et al., 2016].

Las redes neuronales convolucionales están compuestas por diversas capas en donde se encuentran la capa de convolución y una capa de agrupamiento o *pooling*, las cuales serán explicadas posteriormente. Este proceso se repite hasta tener matrices que puedan ser representadas en un vector, que pueda ser computacionalmente introducido como entrada de una red neuronal. A este proceso se le conoce como *flattening* [Eugenio, 2019].

2.1.3.1. Capa de convolución

La convolución es un operador matemático encargado de transformar dos funciones en una tercera, la cual representa la magnitud en la que se superpone f y una traslación invertida de g . Esta operación es denotada por el signo $*$. Este operador es de vital importancia en las redes neuronales convolucionales, ya que el objetivo principal de esta capa es la extracción de características importantes de la señal de entrada en los denominados mapas de características o *feature maps*. Se denota en su forma más general como se muestra en las ecuaciones 2.5 y 2.6. La ecuación 2.5 se utiliza para funciones, mientras que la ecuación 2.6 se emplea para sucesiones y generalmente son aplicadas tanto para el procesamiento de señales como de imágenes.

$$s(t) = (f * g)(t) = \int_{-\infty}^{\infty} f(a) \cdot g(t - a) da \quad (2.5)$$

$$s(t) = (x_n * y_n) = \sum_{k=0}^n x_k y_n - k \quad (2.6)$$

La convolución puede verse también, de manera más abstracta, como el producto de dos matrices donde la primera es la señal de entrada y a la segunda se le denomina kernel o filtro. El resultado de esta operación es el resultado de convolucionar la

señal de entrada con el kernel. Al realizar la convolución de una entrada de $M \times N$, con un kernel o filtro de $A \times B$, se obtendrá una salida de $(M - A + 1)(N - B + 1)$ [Habibi Aghdam and Jahani Heravi, 2017]

2.1.3.2. Capa de pooling

La capa de agrupación o *pooling* en las redes neuronales convolucionales, nos ayudan a reducir la dimensionalidad del mapa de características, obtenido mediante la capa de convolución, haciendo una simplificación de la información y generando una versión condensada de la información contenida en la capa de convolución [Torres, 2020].

Para poder generar esta reducción de dimensionalidad del mapa de características, es necesario contar con un factor denominado *stride*, el cual nos indica cuantas posiciones se moverá el kernel. Por ejemplo, si se tiene un vector de características de 10 elementos (dimensionalidad de diez) y un *stride* de 2, la dimensionalidad resultante del vector será de 5, ya que el kernel se irá moviendo de la posición 0 a la posición 5 de dos en dos y los valores que queden dependen del tipo de *pooling* que se haya realizado, ya sea *Max-Pooling* o *Average Pooling*. Considerando un *stride* igual a dos, *Max-Pooling* recorrerá el vector en pares y eliminará el valor más pequeño, quedándose sólo el valor más alto. En cambio, si se utiliza *Average Pooling*, se calcula la media aritmética de los dos valores y ese será el valor que quedará dentro del vector de características. Por lo tanto, si se utiliza *Max-Pooling*, se obtienen al final los valores mas altos, mientras que con *Average-Pooling*, se obtiene la media de los valores [Eugenio, 2019]. Esto puede apreciarse en las Figuras 2.1 y 2.2.

$$\begin{array}{|c|c|c|c|c|c|} \hline 22 & 10 & 11 & 5 & 5 & 2 \\ \hline \end{array} = \begin{array}{|c|c|c|} \hline 22 & 11 & 5 \\ \hline \end{array}$$

Figura 2.1: Aplicación de la función *Max-Pooling* a un vector de dimensionalidad seis

$$\begin{array}{|c|c|c|c|c|c|} \hline 22 & 10 & 11 & 5 & 5 & 2 \\ \hline \end{array} = \begin{array}{|c|c|c|} \hline 16 & 8 & 3.5 \\ \hline \end{array}$$

Figura 2.2: Aplicación de la función *Average Pooling* a un vector de dimensionalidad seis

2.1.4. Matriz de confusión

Una matriz de confusión es una métrica utilizada para la evaluación de un clasificador, en donde se pueden clasificar dos o mas clases. Es una matriz de $N \times N$ en donde N es el número de clases del problema y se realiza una comparativa entre las predicciones obtenidas y los datos reales. Así mismo, esta matriz se compone de cuatro valores, los cuales se pueden observar en la Figura 2.3: verdaderos positivos, falsos positivos, verdaderos

negativos y falsos negativos (TP, FP, TN y FN, por sus siglas en inglés, respectivamente) [Martinelli, 2022]:

- **Verdaderos positivos (TP):** son los elementos detectados correctamente como positivos.
- **Falsos positivos (FP):** cantidad de elementos positivos que fueron clasificados incorrectamente como positivos.
- **Falsos negativos (FN):** cantidad de elementos negativos que fueron clasificados incorrectamente como negativos.
- **Verdaderos negativos (TN):** cantidad de elementos negativos que fueron clasificados correctamente como negativos.

	PREDICCIÓN	
REAL	TP	FN
	FP	TN

Figura 2.3: Matriz de confusión

Mediante la generación de esta matriz se pueden obtener diversas medidas de desempeño que permiten la evaluación del sistema implementado para la organización de las clases. Estas medidas pueden ser: Precisión (*Precision*), Sensibilidad (*Recall*), Exactitud (*Accuracy*), Especificidad y el valor *F1-Score* [Martinelli, 2022].

2.1.4.1. Precisión

La precisión es la métrica más común para evaluar el desempeño de un modelo. Indica el número de muestras clasificadas correctamente, en comparación con el número total de muestras. Puede calcularse a partir de la matriz de confusión como:

$$\text{Precisión} = \frac{TP}{TP + FP} \quad (2.7)$$

La precisión tiene limitaciones, especialmente en conjuntos de datos no balanceados, donde la exactitud puede ser más alta de lo que realmente es, debido a una gran cantidad de verdaderos negativos. Por lo tanto, la precisión por sí sola puede no ser suficiente para evaluar el rendimiento del modelo.

2.1.4.2. Sensibilidad (*Recall*)

La sensibilidad indica la proporción de casos positivos que fueron correctamente identificados por el modelo. Se calcula como:

$$\text{Sensibilidad} = \frac{TP}{TP + FN} \quad (2.8)$$

La sensibilidad es importante para identificar la capacidad del modelo para encontrar todos los casos positivos en el conjunto de datos.

2.1.4.3. Especificidad

La especificidad representa la proporción de casos negativos que fueron correctamente identificados por el modelo. Se calcula mediante la fórmula:

$$\text{Especificidad} = \frac{TN}{TN + FP} \quad (2.9)$$

La especificidad es crucial para determinar la capacidad del modelo para identificar casos negativos.

2.1.4.4. F1-Score

El F1-Score es la media armónica de precisión y sensibilidad. Ofrece un equilibrio entre ambas métricas y se calcula de la siguiente manera:

$$\text{F1-Score} = 2 \times \frac{\text{Precisión} \times \text{Sensibilidad}}{\text{Precisión} + \text{Sensibilidad}} \quad (2.10)$$

El *F1-Score* es útil cuando hay una distribución desigual entre las clases en el conjunto de datos.

2.1.5. Protocolo de comunicación HTTP

El Protocolo de Transferencia de Hipertexto (*Hypertext Transfer Protocol*) es un protocolo cliente-servidor, de capa de aplicación, que realiza intercambios de información entre clientes Web y servidores HTTP [León, 2019], en donde el cliente Web es el que solicita, recibe y visualiza objetos Web, mientras que el servidor HTTP envía los objetos en respuesta a las peticiones del cliente Web.

HTTP es un protocolo sin estado, lo que significa que cada solicitud que se procesa, se hace de manera independiente, i.e, no se tiene conocimiento de las solicitudes anteriores. Sin embargo, se puede gestionar el estado por otro tipo de vías (e.g., *cookies* o sesiones). Aunado a esto, este protocolo de comunicación sigue un modelo de solicitud-respuesta en

donde se inicia una conexión, se envía una solicitud, se recibe respuesta y finalmente se cierra la conexión.

2.1.5.1. Componentes de una solicitud HTTP

Una solicitud HTTP consiste en una línea de solicitud que contiene [Gourley et al., 2002]:

- **Método:** GET, POST, PUT, DELETE, etc.
- **URL:** dirección del recurso solicitado.
- **Versión del protocolo:** HTTP/1.1, HTTP/2, etc.
- **Encabezados (*Headers*):** información adicional sobre la solicitud, como tipo de contenido aceptado, información del agente de usuario, tipo de compresión, etc.
- **Cuerpo de la solicitud (*Body*):** este elemento es opcional y puede contener datos enviados al servidor, como formularios en una solicitud POST.

2.1.5.2. Componentes de una respuesta HTTP

Una respuesta HTTP consta de una línea de estado que contiene [Gourley et al., 2002]:

- **Código de estado:** indica el éxito o el tipo de error que ha ocurrido (e.g., 200 OK, 404 *Not Found*, 500 *Internal Server Error*).
- **Encabezados (*Headers*):** proporcionan información adicional sobre la respuesta, como tipo de contenido, longitud del contenido, *cookies*, etc.
- **Cuerpo de la respuesta (*Body*):** este elemento, al igual que en la solicitud, es opcional. Contiene los datos solicitados (e.g., página HTML, una imagen, un archivo de texto, etc).

2.1.5.3. Métodos para el envío de solicitud

- **GET**
 - **Propósito:** solicita la representación de un recurso específico.
 - **Características:** es un método que no tiene efectos secundarios en el servidor y puede repetirse sin cambiar el estado del servidor o del recurso.
- **POST**
 - **Propósito:** envía datos al servidor para crear un nuevo recurso.

- **Características:** puede generar cambios en el servidor cada vez que se envía. Es comúnmente utilizado para enviar datos de formularios, realizar operaciones de inicio de sesión, enviar archivos, etc.

- **PUT**

- **Propósito:** actualiza un recurso existente o crea uno nuevo si no existe.
- **Características:** si se realiza la misma operación varias veces, no debería cambiar el resultado después de la primera solicitud exitosa.

- **DELETE**

- **Propósito:** elimina el recurso especificado por la URL
- **Características:** al igual que el método POST, es capaz de realizar cambios en el servidor.

- **PATCH**

- **Propósito:** realiza modificaciones parciales a un recurso
- **Características:** es un método similar al método PUT, sin embargo, se utiliza para aplicar cambios parciales a un recurso en lugar de reemplazarlo por completo.

2.1.5.4. Versiones HTTP

- **HTTP/1.0:**

- esta es la primera versión del protocolo HTTP.
- Cada solicitud establece una conexión TCP separada.

- **HTTP/1.1:**

- se introdujeron mejoras como conexiones persistentes (*keep-alive*) para reducir la sobrecarga de la conexión y el tiempo de carga, así como la reutilización de conexiones para varias solicitudes.

- **HTTP/2:**

- Mejora la velocidad y eficiencia de la Web.
- Introduce multiplexación y comprensión de encabezados, y permite la transmisión de datos en paralelo sobre una sola conexión.

2.2. Trabajos relacionados

Dentro de los avances tecnológicos que se han desarrollado a través del tiempo, se encuentran aquellos que están relacionados con el poder dotar a las computadoras con la capacidad de ser utilizadas para analizar y entender el habla humana [Latif et al., 2021], con el fin de poder facilitar la interacción con las personas o con otras máquinas.

El estudio del procesamiento del lenguaje natural (PLN) se ha llevado a cabo durante años y, debido a su complejidad, se ha tenido la necesidad de generar sub-áreas que permitan enfocarse, de manera más específica, en ciertas características que conforman el habla humana. Un ejemplo de esto son las sub-áreas conocidas como *speech recognition (SR)* y *voice recognition (VR)*, que si bien pudieran parecer lo mismo, dentro de las áreas de las Ciencias de la Computación y de la Ingeniería Eléctrica, no lo son. La primera, también conocida como *Automatic Speech Recognition (ASR)* o *Speech to Text (STT)* [IBM, 2020], se enfoca en la capacidad que tiene un programa de procesar y transformar el habla humana en un formato de escritura; mientras que la segunda está enfocada en la identificación de la persona que habla, pero no en lo que dice [Renjith and Manju, 2017].

A continuación, se describen algunos trabajos relevantes en el dominio de los asistentes digitales y su evaluación dentro del sector del cuidado de la salud.

En el trabajo de [Dojchinovski et al., 2019], se desarrolló un asistente personal comandado por voz, haciendo uso de los servicios de los asistentes personales *Alexa* y *Google Assistant*. Este sistema tiene el propósito de ayudar a un paciente en el monitoreo de sus signos vitales (cardíacos) dentro del hogar y en el control de diversas actividades, e.g., agenda de citas médicas, recopilación de la terapia e ingesta de medicamentos. Para ello, el sistema hace notificaciones y recordatorios al paciente cuando éste necesita realizar alguna acción relacionada con su salud.

El asistente personal *Alexa*, utilizado en esta propuesta, cuenta con una serie de micrófonos de alta sensibilidad y requiere de acceso a Internet para poder utilizar el sistema de procesamiento de lenguaje natural de Amazon. Además, solamente se tiene interacción con este asistente personal cuando se le pide hacerlo por medio de palabras clave (e.g., *Alexa*).

En el caso de *Google Assistant*, éste utiliza *Google Action* como contraparte de las capacidades que se tienen en *Alexa* para poder extender sus funcionalidades. *Google Action* está constituido de tres partes principales: intentos, acciones y cumplimiento.

Los componentes de la arquitectura de este asistente son los siguientes:

- Sensor Electrocardiograma (ECG)

- Aplicación que comunica con el sensor ECG
- Comunicación Bluetooth
- Almacenamiento en la nube
- Asistente *Alexa* y *Google Assistant*

Los servicios que ofrece este asistente son:

- Verificación de la condición cardíaca
- Envío al médico de un informe de la condición cardíaca del paciente
- Programación de citas
- Recordatorio de terapia mediante notificaciones de voz

Por otra parte, [Salai et al., 2021] desarrollaron *IntraVox*, un sistema basado en una interacción vocal para brindar soporte a los usuarios con necesidades complejas, como aquellos que sufren demencia o autismo. Este sistema integra un comando de voz en los dispositivos domésticos inteligentes y sensores, con el fin de mejorar la calidad de vida de este tipo de personas.

Este sistema se sustenta con la premisa de que, a pesar de que los asistentes personales comandados por voz se están volviendo populares, existe un grupo de personas que puede tener problemas para interactuar con estos asistentes, como las personas con autismo o con problemas del habla. *IntraVox* está diseñado para ayudar a este tipo de personas, enviando comandos de voz a un asistente como *Alexa* o *Google Assistant*, con base en los datos recolectados por los sensores que se encuentran en el hogar.

Los componentes de la arquitectura de este sistema son los siguientes:

- Raspberry Pi4B.
- Sensores para la detección de las necesidades del paciente:
 1. Movimiento
 2. Temperatura
 3. Sonido
 4. Luz
- Bocina para el envío de comandos de voz al asistente.

- Asistente, como *Alexa*, para la ejecución de comandos de voz.

Dos de los problemas que se presentaron durante el desarrollo de este sistema se enuncian a continuación: 1) al aplicar técnicas de procesamiento de lenguaje natural para realizar la conversión de texto a audio, con el fin de dar una instrucción al asistente digital mediante una bocina, no se lograba que el mensaje fuera claro al momento de reproducir el audio, provocando que el asistente *Alexa* no comprendiera la instrucción que se le daba, por lo cual, se tomó la decisión de precargar audios que dieran la instrucción, dependiendo de la lectura de los sensores; y 2) al querer dar dos instrucciones dentro de una oración, *Alexa* sólo reconocía una de las dos, por lo cual, se decidió utilizar únicamente palabras clave para dar los comandos.

IntraBox hace uso del lenguaje de programación Python, ya que mediante este lenguaje se manipulan los sensores utilizados para la percepción del ambiente en donde se encuentra el usuario. Además, se pueden realizar ajustes a los parámetros de los sensores como los grados centígrados (para la temperatura), decibeles (para el audio) y lumen (para luz); todos estos ajustes se pueden realizar de manera ágil, ya que sólo es cuestión de cambiar los valores dentro del código elaborado en Python.

[Rehman et al., 2020] desarrollaron un asistente en tiempo real instruido médicamente (*MIRA*, por sus siglas en inglés). Este asistente se basa en escuchar el síntoma principal del usuario y, con base en esto, predice entre dos posibles enfermedades (diabetes y glaucoma) cuál es la que más probabilidad tiene el usuario de padecer.

Para la implementación de la arquitectura de este asistente, se recolectaron los principales síntomas con los corpus de diálogo de pacientes reales, se validaron manualmente los datos bajo la supervisión de médicos especialistas, se utilizaron los datos para el entendimiento del lenguaje natural (NLU, por sus siglas en inglés) y se realizó la identificación de la enfermedad para la generación de una respuesta.

Los componentes de la arquitectura del asistente son los siguientes:

- Módulo de reconocimiento de habla (*speech recognition*).
- Módulo de NLU
- Submódulo de registro de identidad
- Submódulo de verificación de identidad
- Manejador de diálogo:
 1. Datos de historia

2. Estado de seguimiento
3. Plantillas de diálogo
4. Datos principales de la queja
5. Historial médico
6. Formulación de respuesta

Para la construcción del prototipo, se eligió el sistema operativo Android debido a la compatibilidad que tiene actualmente. Así mismo, para el módulo de “Reconocimiento del habla”, se utilizó un algoritmo basado en redes neuronales. El módulo de NLU determina la naturaleza de la intención de la entrada del usuario, en función del modelo entrenado y reenvía una solicitud a un módulo específico.

Al evaluar el prototipo sólo tres participantes pudieron interactuar simultáneamente en su diseño experimental, por lo que se crearon 11 grupos de tres personas cada uno. El resultado final de las evaluaciones se muestra en la Tabla 2.2.

Tabla 2.2: Evaluación del sistema reportado por [Rehman et al., 2020]

Medida	Glucosa	Diabetes
Presición	90.90 %	84.80 %
Recall	90.91 %	84.085 %
Sensitividad	93.94 %	96.97 %
F-score	89.55 %	88.89 %

En el trabajo de [Jesus-Azabal et al., 2020], se implementó un asistente de voz enfocado en adultos mayores, ya que existe una gran cantidad de este tipo de personas en Europa y se espera que, en las siguientes décadas, el número aumente. Este asistente tiene como principal objetivo poder ayudar a las personas a recordar su tratamiento médico, sin la necesidad de que dicho asistente se encuentre conectado a Internet. Por esta razón, se pretende que el asistente sea también de utilidad a personas que viven en zonas rurales en donde no se tiene conexión a Internet.

El asistente cuenta con una plataforma, la cual brinda al personal médico la posibilidad de poder agregar prescripciones e incluso crear una cita para poder recordarle al usuario y no la pierda. Para poder llevar el control del usuario, la información de éste debe ser ingresada a la plataforma para poder generarle una cuenta y sobre ella poder crear los

recordatorios o prescripciones del médico.

La arquitectura de este asistente consta de una aplicación para *smartphone* desarrollada en el lenguaje *Kotlin*. Esta aplicación cuenta con un modelo de navegación de pestañas, así mismo genera un *payload* JSON con la información del usuario para después poder realizar una sincronización vía Bluetooth con el dispositivo *snip*. Este dispositivo es un asistente de voz *open source* que no necesita de conexión a Internet para poder funcionar adecuadamente; además, cuenta con un componente de procesamiento de lenguaje natural que permite poder entender las peticiones del usuario con base en palabras clave, las cuales son especificadas utilizando el lenguaje de programación Python. Así mismo, se utilizan bibliotecas de calendarización para poder programar los recordatorios.

Dentro de los trabajos que se han llevado a cabo para el apoyo a personas con diabetes, se encuentra *VoiceDiab* [Foltynski et al., 2018]. Este sistema está diseñado para ayudar a personas con diabetes de tipo uno a realizar el cálculo de sus dosis de insulina, con el fin de compensar un alimento. *VoiceDiab* es un sistema cliente/servidor que recibe por voz la descripción de los alimentos a compensar mediante un *smartphone* con sistema operativo Android. A través de este dispositivo, se ofrece una aplicación que se comunica con un conjunto de tres servidores responsables de:

1. *Automatic Speech Recognition*.
2. Análisis de la descripción textual de los alimentos para la determinación de su composición.
3. Cálculo de la dosis de insulina apropiada para la compensación de los alimentos.

El sistema está constituido por una base de datos de 887 productos de comida básicos, con la información de su contenido calórico, carbohidratos, etc. Aunado a esto, el sistema tiene un diccionario de idiomas correspondiente a la base de datos de los alimentos y una lista de unidades de medida, e.g., una cucharada, 20 gramos, una taza, etc. para poder entender, de manera correcta, las porciones de alimentos que se le ingresan.

Para utilizar este sistema, se detecta primero la cantidad de ruido que hay en el ambiente y si éste es aceptable aparecerá el mensaje "Habla por favor" en el *smartphone*. Una vez que se comienzan a decir los alimentos al sistema, toma unos 15 segundos aproximadamente en desplegar la lista de alimentos que se han ingresado para corroborar, eliminar o agregar algún alimento; finalmente, ya que se tienen los alimentos correctos, se realiza el cálculo de la dosis de insulina, el cual toma alrededor de 2 a 3 segundos en realizarse y mostrarse en el *smartphone* junto con la cantidad de carbohidratos, proteínas, etc. que contienen los alimentos ingresados.

Los resultados obtenidos al evaluar este sistema son los siguientes:

- El 40 % de los participantes calificaron la utilidad del sistema con diez, en una escala de 0 a 10 y el 70 % de los participantes lo calificaron con una puntuación mayor a cinco.
- Los pacientes expresaron la necesidad de corregir la descripción de los alimentos, ya que el 49 % de los productos que ellos ingresaron no se encontraban en la base de datos.
- Mas del 87 % de los participantes estarían dispuestos a utilizar el sistema en un futuro.
- El 81 % admitió que la dosis calculada manualmente difería de las obtenidas por el sistema.

Otro de los trabajos relacionados con la atención a personas que padecen diabetes es el desarrollado por [Cheng et al., 2018]. Este sistema está enfocado en reducir las complicaciones que se pueden presentar en personas mayores que sufren de diabetes tipo dos, al utilizar aplicaciones móviles de la salud. El sistema propone emplear una combinación entre un asistente digital (*Google Home*) para alojar al agente conversacional y una interfaz web para la visualización de datos y así poder asistir, de una mejor manera, a las personas mayores con diabetes tipo dos.

El sistema se centra en cinco aspectos, los cuales son:

- Afrontamiento saludable
- Monitoreo
- Toma de medicamentos
- Comida saludable
- Actividad física

Los componentes de la arquitectura del sistema son los siguientes:

- Asistente *Google Home*
- Webhook
- Base de datos
- Interfaz web

Google Home es el núcleo de todo el sistema, debido a que la interacción se da mediante este asistente que utiliza entendimiento del lenguaje natural (NLU, por sus siglas en inglés) y *Machine Learning* para la comprensión de las instrucciones que se le dan. El algoritmo de *Machine Learning* que utiliza este asistente tiene la capacidad de reentrenarse para evitar fallas en el reconocimiento de nuevos comandos de voz.

Webhook se generó, utilizando Node.js y el paquete Express Node, y es el encargado de leer constantemente respuestas JSON provenientes de API.AI, la cual contiene parámetros de entrada en lenguaje natural. Este agente cuenta con respuestas predeterminadas y almacenadas en su base de datos. Por otra parte, la interfaz web está diseñada para que el usuario pueda monitorear su estatus mental y de salud, mediante una representación gráfica interactiva. Dicha gráfica se genera en tiempo real mientras el usuario habla con el asistente.

A continuación se muestra en la Figura 2.4 un esquema de la arquitectura del sistema.

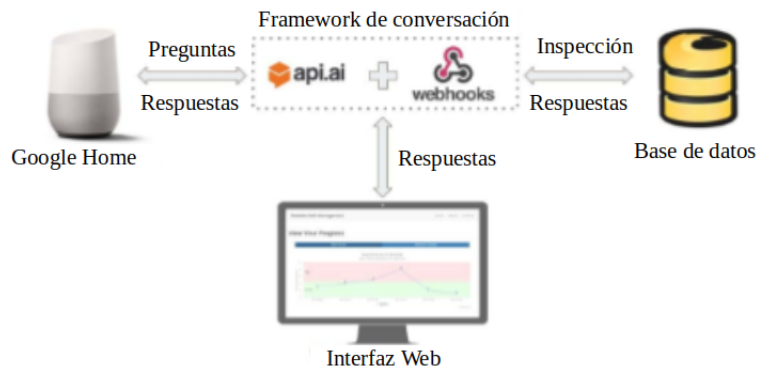


Figura 2.4: Arquitectura del sistema mostrando la organización direccional de entradas y salidas de datos [Cheng et al., 2018]

Dentro del trabajo relacionado, se encuentran estudios acerca de la eficiencia, precisión y calidad que tienen los asistentes digitales comerciales en la actualidad como *Alexa*, *Google Assistant* y *Siri*, en el ámbito del cuidado de la salud.

En el trabajo de [Alagha and Helbing, 2019] se reportan los resultados de un estudio realizado a los asistentes mencionados sobre el tema de seguridad y uso de vacunas. La evaluación constó de un cuestionario de 54 preguntas, las cuales fueron seleccionadas de los sitios de preguntas frecuentes (FAQ, por sus siglas en inglés) de las páginas gubernamentales e.g, Servicio de Salud Nacional (NHS, por sus siglas en inglés), CDC, etc. y consultas de búsqueda orgánica en la Web sobre vacunas. La selección de las preguntas se llevó a cabo de este modo para asegurarse de que las preguntas realizadas reflejaran tanto la experiencia de las agencias gubernamentales como casos de uso realistas de buscadores

de información en línea.

Para la evaluación de las respuestas dadas por los asistentes digitales, se elaboró una rúbrica que asignaba puntos con base en la calidad de las fuentes citadas, la precisión de la respuesta dada y el experiencia de los autores. Así mismo, los estándares de calidad de la rubrica generada se basaron en el criterio del *benchmark* JAMA para la evaluación de páginas web, así como la jerarquía de información/consejos de salud creada por Boyd y Wilson.

Los resultados obtenidos de esta prueba se pueden observar en la Tabla 2.3, donde la puntuación media se encuentra en un rango de 0 a 6, siendo 0 que el asistente no entendió la pregunta o no dio una respuesta y 6 que el asistente respondió de manera correcta y se basó en evidencia proveniente de páginas gubernamentales.

Tabla 2.3: Resultados obtenidos por [Alagha and Helbing, 2019] sobre la evaluación de asistentes digitales

Asistente	<i>Alexa</i>	<i>Google Assistant</i>	<i>Siri</i>
Puntuación media	0.98	5.10	5.16
Se entendió la pregunta y se respondió	25 %	94 %	94 %
Promedio de palabras habladas por respuesta	29	21	13
Recursos mas citados	Wikipedia.org	CDC.gov	CDC.gov

Como conclusiones a las que se llegaron en este trabajo son:

- *Google Assistant* y *Siri* entienden con buena precisión las peticiones que se les dan y brindan enlaces a recursos confiables acerca de las vacunas.
- El rendimiento de *Alexa* es pobre en el entendimiento de peticiones por voz y no proporciona respuestas de calidad, sin embargo, *Alexa* intentó entregar respuestas de sólo audio con más frecuencia que los otros asistentes.

En artículos relacionados, citados en este trabajo, se reportaron algunas otras deficiencias en los asistentes, como se menciona a continuación:

- [Miner et al., 2016] reportaron que existen inconsistencias por parte de Google y Apple, al momento de resolver preguntas acerca de la salud mental y violencia interpersonal.

- [Boyd and Wilson, 2018] menciona que la información sobre dejar de fumar es poca, utilizando *Siri* y *Google Assistant*.
- [Wilson et al., 2017] obtuvo que las respuestas sobre salud sexual basada en fuentes de expertos son sólo de un 48 %.

A continuación en la Tabla 2.4 se muestra una comparativa de los trabajos reportados anteriormente.

Tabla 2.4: Comparativa de los trabajos reportados en el estado del arte

Trabajo	Enfermedad	Lenguaje	Asistente	Servicios
Foltynski et al., 2018	Diabetes tipo 1	-	Cualquier dispositivo con sistema operativo Android	<ul style="list-style-type: none"> ▪ Despliegue de los alimentos ingresados mediante voz por el usuario. ▪ Cálculo y despliegue de las calorías totales de los alimentos ingresados. ▪ Cálculo de dosis de insulina que el usuario debe administrar.
Cheng et al., 2018	Diabetes tipo 2	Node.js	<i>Google Home</i>	<ul style="list-style-type: none"> ▪ Monitoreo del paciente. ▪ Recordatorio de toma de medicamentos. ▪ Comida saludable.

Dojchinovski et al., 2019	Problemas Cardiacos	-	<i>Alexa y Google Assistant</i>	<ul style="list-style-type: none"> ▪ Verificación de la condición cardíaca. ▪ Envío de informe de la condición cardíaca del paciente al médico. ▪ Calendarización de citas. ▪ Recordatorios de terapia.
Rehman et al., 2020	Diabetes o Glaucoma	-	<i>MIRA</i>	<ul style="list-style-type: none"> ▪ Servicio de identificación del usuario mediante reconocimiento de voz. ▪ Identificación entre dos posibles enfermedades con base en los síntomas dichos por el usuario.
Jesus-Azabal et al., 2020	Cualquiera	<i>Kotlin</i> <i>Python</i>	<i>snip</i>	<ul style="list-style-type: none"> ▪ Recordatorio de toma de medicamentos a personas mayores. ▪ Seguimiento médico del usuario

Salai et al., 2021	Autismo o Demencia	Python	<i>Alexa o Google Assistant</i>	<ul style="list-style-type: none">▪ Detección de necesidades por medio de sensores.▪ Comunicación con el asistente digital.
--------------------	--------------------	--------	---------------------------------	--

Capítulo 3

Análisis y diseño del sistema de reconocimiento de voz y habla multi-fuente

En este capítulo se explica el diseño del sistema de reconocimiento de voz y habla multi-fuente, el cual está constituido por tres sistemas: 1) aplicación Android, 2) sistema de captura permanente y 4) micro-servicio. En la Sección 3.1 se explica el diseño de la aplicación desarrollada para dispositivos Android. La Sección 3.2 detalla el diseño del sistema de captura permanente para la interacción con los usuarios. Finalmente, la Sección 3.3 trata del diseño del micro-servicio elaborado para el procesamiento de las peticiones de usuarios realizadas mediante la aplicación Android y los dispositivos de captura permanente.

3.1. Aplicación Android

En esta sección se presenta una descripción de la aplicación para *smartphones* con sistema operativo Android, seguido de los requerimientos funcionales, requerimientos no funcionales y las clases a utilizar para el diseño de la aplicación.

3.1.1. Funcionamiento de la aplicación

El objetivo principal de la aplicación es generar un modelo de comunicación cliente-servidor para una arquitectura de micro-servicios, que permita el envío de peticiones de un usuario a un micro-servicio, haciendo uso del protocolo de comunicación HTTP. Dichas peticiones son capturadas mediante el micrófono del *smartphone* en un formato de archivo de audio (mp3 o mp4). La aplicación sólo puede capturar y enviar una petición a la vez.

Para la captura de la petición del usuario, la aplicación contiene un botón colocado en la parte central inferior de la pantalla. Dentro de las acciones de este botón se encuentran, habilitar el micrófono del *smartphone* y comenzar la captura de la petición del usuario, así como generar el archivo de audio que contiene dicha petición. Este archivo de audio es eliminado una vez que la petición ha sido enviada o cuando se tiene algún problema con él.

Al presionar el botón para comenzar a grabar la petición, se muestra un aviso de corta duración para informar al usuario que se ha comenzado a grabar su petición. Para terminar de grabar la petición y enviarla al micro-servicio, el usuario debe volver a presionar el botón mencionado. Una vez que se haya presionado el botón por segunda vez para terminar de grabar la petición, se muestra un aviso de corta duración, indicando que se ha completado la grabación de la petición.

La aplicación contiene un cuadro de texto para mostrar al usuario las respuestas del micro-servicio, así como los pasos a seguir para poder realizar el registro de un nuevo usuario. El tamaño de los mensajes puede ser ajustado por el usuario siempre y cuando el tamaño permita desplegar de manera apropiada el mensaje dentro del cuadro de texto. De manera adicional, se tiene un botón colocado por debajo del cuadro de texto que brinda la posibilidad al usuario de reproducir el texto desplegado, utilizando voz sintética, mediante la bocina del *smartphone*.

La aplicación cuenta con un botón colocado en la parte superior izquierda que, al ser presionado, despliega un menú en donde se tienen diferentes opciones, dentro de las cuales se encuentran 1) agregar un nuevo usuario, 2) configurar y 3) cerrar la aplicación. Al presionar la primera opción, se inicia el proceso para agregar un nuevo usuario al micro-servicio. Las instrucciones a seguir para realizar este proceso son desplegadas en el cuadro de texto. Dentro de los pasos a seguir, se requiere el envío de muestras de audio al micro-servicio (encargado de la generación del modelo para el reconocimiento de voz) para su posterior procesamiento y el registro del usuario. Las muestras requeridas son capturadas y enviadas mediante el botón inferior central anteriormente mencionado. Una vez finalizado el proceso de registro de un nuevo usuario, se puede seguir utilizando la aplicación para el envío de peticiones al micro-servicio.

3.1.2. Requerimientos

A continuación, se muestran los requerimientos funcionales y no funcionales correspondientes al diseño de la aplicación Android.

3.1.2.1. Requerimientos funcionales

1. La aplicación recibirá la petición del usuario, mediante el micrófono del *smartphone*, al presionar una vez el botón colocado en la parte inferior central y generará un archivo de audio.
2. La aplicación enviará al micro-servicio un archivo de audio que contenga la petición del usuario. Dicha acción se realizará cuando el usuario presione por segunda vez el botón colocado en la parte inferior central de la aplicación.
3. La aplicación contendrá un cuadro de texto que permita desplegar las respuestas generadas por el micro-servicio e instrucciones a seguir en caso de que se requiera.
4. El usuario podrá solicitar que la aplicación lea, mediante voz sintética, los mensajes mostrados en el cuadro de texto al presionar el botón localizado en la parte inferior izquierda de dicho cuadro de texto.
5. Un usuario podrá iniciar su proceso de registro en el micro-servicio al presionar el botón colocado en la parte superior izquierda de la aplicación.

3.1.2.2. Requerimientos no funcionales

1. La aplicación sólo podrá generar y enviar un archivo de audio a la vez.
2. El archivo de audio generado podrá ser .mp3 o .mp4.
3. El protocolo de comunicación entre la aplicación y el micro-servicio será HTTP.
4. La aplicación verificará que exista conexión con el micro-servicio.
5. La aplicación no requerirá datos personales del usuario; para su registro sólo se utilizarán las muestras de voz.
6. El usuario deberá generar y enviar al micro-servicio muestras de audio con una duración de 40 segundos para registrar a un nuevo usuario.
7. La aplicación contendrá un apartado de configuración que permita al usuario modificar el tamaño de letra de los mensajes desplegados en el cuadro de texto, así como el formato en el que desea guardar el archivo de audio.
8. La aplicación deberá generar una bitácora en donde se reporten inconvenientes durante su uso, para su posterior análisis.

3.1.3. Casos de uso

A continuación, se abordan los temas relacionados con los casos de uso de la aplicación Android. Primeramente, se mencionan los actores involucrados y sus actividades. En seguida, se presentan las especificaciones de cada caso de uso.

3.1.3.1. Actores y sus actividades

- Usuario:
 - Generar una petición y enviarla al micro-servicio.
 - Leer los mensajes mostrados en el cuadro de texto o solicitar a la aplicación que los lea mediante voz sintética.
- Micro-servicio:
 - Recibir peticiones de la aplicación.
 - Enviar respuestas de las peticiones a la aplicación.

3.1.3.2. Especificaciones de casos de uso

En la Figura 3.1, se muestra el diagrama de casos de uso referente a la aplicación Android para la comunicación y envío de peticiones, mediante archivos de voz, al micro-servicio.

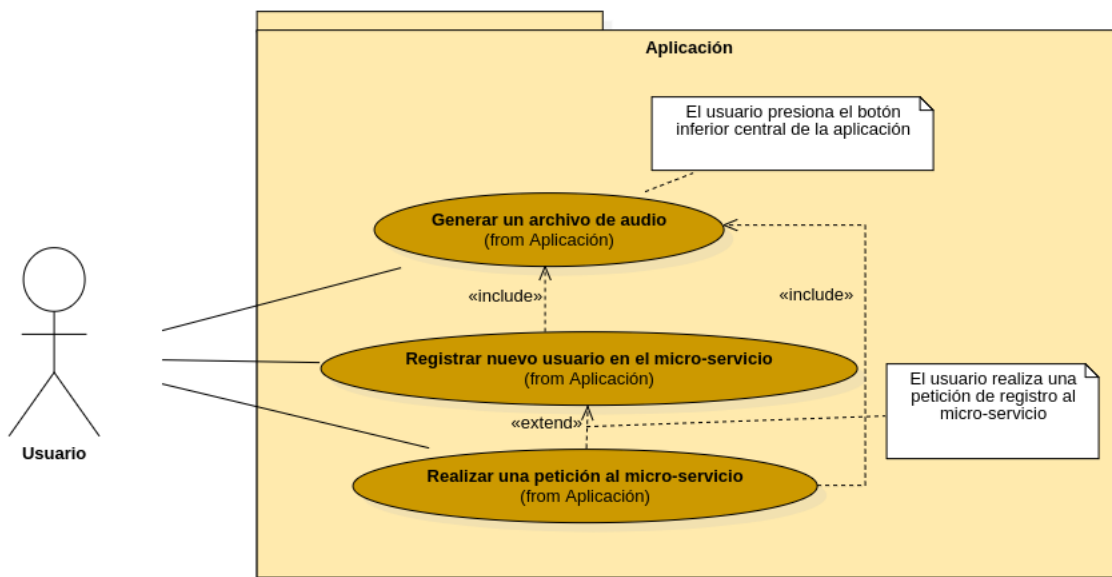


Figura 3.1: Diagrama de casos de uso de la aplicación Android.

A continuación, se especifica cada uno de los casos de uso de la aplicación Android, indicando el actor principal, el objetivo del caso de uso, las precondiciones, el disparador, un escenario, la prioridad, la frecuencia de uso, las excepciones y los actores secundarios.

Caso de uso	Registrar a un nuevo usuario en el micro-servicio
Actor principal	Usuario
Objetivo	Generar muestras de audio de un usuario no registrado y enviarlas al micro-servicio para su registro.
Precondiciones	<ul style="list-style-type: none"> ■ No se tiene un registro del usuario en el micro-servicio. ■ No se ha podido identificar al usuario previamente.
Disparador	Una persona no registrada en el micro-servicio requiere registrarse para poder ser identificada por este.
Escenario	<ol style="list-style-type: none"> 1. El usuario presiona el botón colocado en la parte superior izquierda para comenzar el registro de usuario. 2. La aplicación despliega un mensaje en el cuadro de texto donde se indican los pasos a seguir para el registro del usuario no registrado. 3. El usuario lleva a cabo el caso de uso Generar un archivo de audio. 4. El usuario lleva a cabo el caso de uso Realizar una petición al micro-servicio. 5. Se repiten las acciones 2, 3 y 4 consecutivamente seis veces más para la generación y el envío de siete muestras de audio. 6. La aplicación despliega un mensaje en el cuadro de texto donde se indica que el registro del usuario se está llevando a cabo. 7. La aplicación recibe la respuesta del micro-servicio y se despliega en el cuadro de texto, indicando que el registro ha sido completado exitosamente.

Prioridad	Alta
Frecuencia de uso	Baja
Excepciones	Si no se tiene una conexión Wi-Fi, la aplicación despliega un mensaje en donde le indica al usuario que debe conectarse a una red Wi-Fi.
Actores secundarios	No

Caso de uso	Generar un archivo de audio
Actor principal	Usuario
Objetivo	Generar un archivo de audio que contenga la petición del usuario o las muestras de audio, en el caso de un registro de usuario.
Precondiciones	No (no se coloca una precondición, ya que cualquier persona puede generar un archivo de audio y enviarlo al micro-servicio. El encargado de identificar si se tiene o no registro de la persona y atender o no la petición es el micro-servicio, no la aplicación. De igual forma, no se requiere una conexión para poder generar el archivo de audio.)
Disparador	El usuario requiere enviar una petición o muestras de audio al micro-servicio.
Escenario	<ol style="list-style-type: none"> 1. El usuario presiona el botón colocado en la parte inferior central para iniciar la grabación del audio. 2. El usuario comienza a externar su petición utilizando el micrófono del <i>smartphone</i>. 3. El usuario presiona por segunda vez el botón inferior central para terminar la grabación del archivo de audio. 4. La aplicación guarda el archivo de audio en la memoria interna del <i>smartphone</i>.
Prioridad	Alta
Frecuencia de uso	Alta

Flujo alternativo	<p>3.1(a) Si el usuario no ha presionado el botón por segunda vez para detener la grabación después de diez segundos, la grabación se detendrá sin guardarse en la memoria del <i>smartphone</i> y se le pedirá que realice de nuevo su petición.</p> <p>3.2(a) El usuario realiza de nuevo su petición presionando, por segunda vez, el botón mencionado y continua con el punto cuatro del escenario.</p>
Excepciones	Ninguna
Actores secundarios	Micro-servicio

Caso de uso	Realizar una petición al micro-servicio
Actor principal	Usuario
Objetivo	Enviar una petición mediante un archivo de audio y recibir respuesta del micro-servicio.
Precondiciones	Se ha generado un archivo de audio.
Disparador	El usuario requiere realizar una petición al micro-servicio. El usuario presiona, por segunda vez, el botón colocado en la parte inferior central.

Escenario	<ol style="list-style-type: none"> 1. El usuario presiona el botón colocado en la parte inferior central por segunda vez para terminar la grabación del archivo de audio. 2. La aplicación verifica que exista una conexión con el micro-servicio mediante el protocolo HTTP. 3. La aplicación envía el archivo de audio al micro-servicio. 4. La aplicación verifica que el micro-servicio ha recibido la petición exitosamente. 5. El micro-servicio atiende la petición solicitada. 6. La aplicación recibe la respuesta del micro-servicio. 7. La aplicación despliega la respuesta del micro-servicio en el cuadro de texto. 8. La aplicación finaliza la conexión con el micro-servicio. 9. El usuario lee el mensaje desplegado en la pantalla del <i>smartphone</i>.
Flujos alternativos	<p>8.1(a) El usuario no logra ver de manera apropiada el mensaje desplegado y realiza un ajuste del tamaño de letra para continuar con el punto nueve del escenario.</p> <p>8.1(b) El usuario no desea o no puede leer el mensaje desplegado y presiona el botón que habilita la lectura del mensaje por parte del <i>smartphone</i> para terminar de manera exitosa el escenario.</p>
Prioridad	Alta
Frecuencia de uso	Alta

Excepciones	<ul style="list-style-type: none"> * Si no se tiene una conexión con el micro-servicio o se pierde, la aplicación mostrará al usuario un aviso de corta duración de error de conexión. * Si el micro-servicio no recibió la petición exitosamente, la aplicación pedirá al usuario volver generar su petición. * Si no se tiene una respuesta por parte del micro-servicio en un tiempo definido, la aplicación cerrará la conexión con el micro-servicio y solicitará al usuario volver a realizar su petición.
Actores secundarios	Micro-servicio

3.1.4. Clases de la aplicación Android

En esta sección se presentan primeramente las clases e interfaces creadas para el diseño de la aplicación Android (ver Sección 3.1.4.1). A continuación se explican las principales funcionalidades de dicha aplicación con base en las clases e interfaces, como la generación de archivos de audio (ver Sección 3.1.4.2), la creación y envío de peticiones (ver Sección 3.1.4.3) y el despliegue y la lectura de mensajes (ver Sección 3.1.4.4). Finalmente, se explica la clase principal de la aplicación Android (ver Sección 3.1.4.5).

3.1.4.1. Clases e Interfaces

Se cuenta con un total de 13 clases y tres interfaces para el diseño de la aplicación en *Android Studio*, las cuales se puede observar en el diagrama de clases de la Figura 3.2. Las clases son las siguientes: AudioMP3, AudioMP4, CreateAudioFileFactory, Log, NewUserPostRequest, UserPostRequest, CreatePostFactory, Subject, PostRequest, Wi-FiConnection, Text, Reader y MainActivity. Las interfaces definidas son: AudioRecorder, Request y Observer.

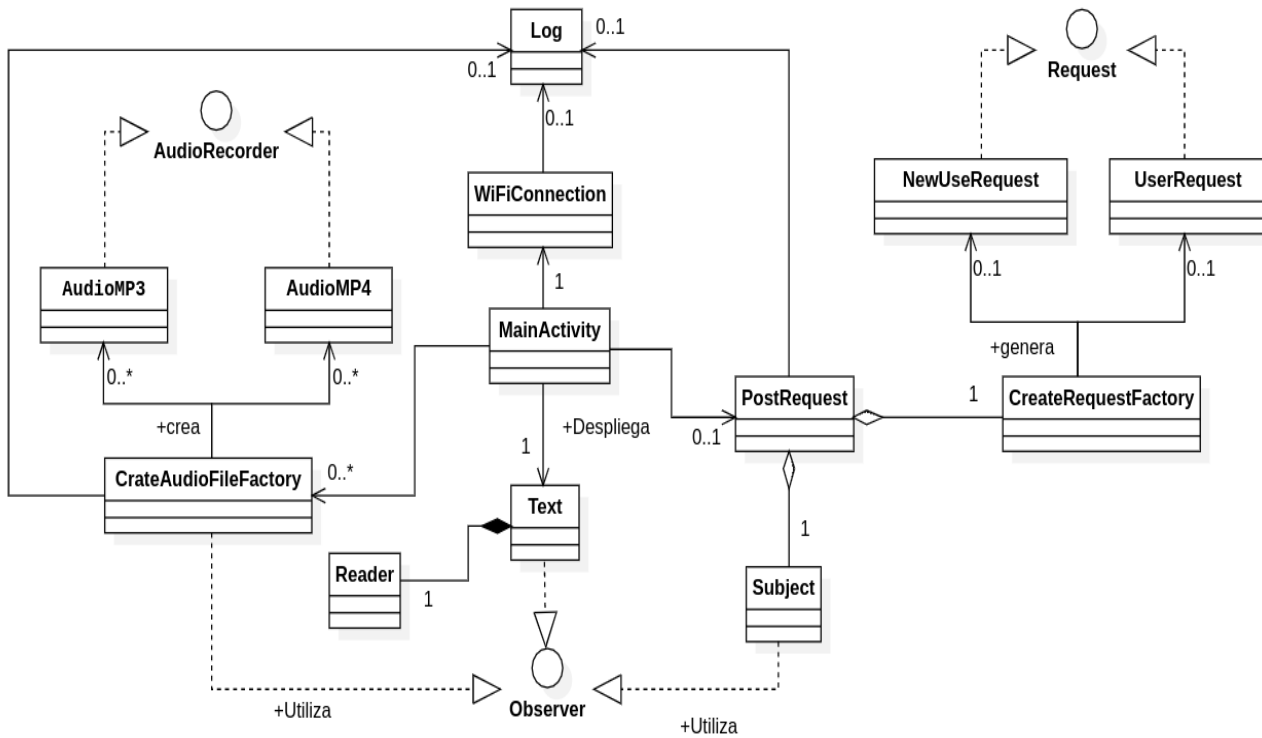


Figura 3.2: Diagrama de clases de aplicación Android

3.1.4.2. Generación de archivos de audio

Para la creación de los archivos de audio que son requeridos para el envío de peticiones de registro de usuario o alguna otra petición realizada al micro-servicio, se utiliza el patrón de diseño *Factory* [Gamma, 2002], el cual permite crear instancias de archivos de audio sin tener la necesidad de especificar su clase exacta.

Para el uso de este patrón de diseño, se generan tres clases: `AudioMP3`, `AudioMP4` y `CreateAudioFactory`. Las dos primeras utilizan la interfaz `AudioRecorder`, la cual contiene los métodos necesarios como `startRecord()`, `stopRecord()`, `saveRecord()` y `deleteRecord()`, para la generación y manejo apropiado de los archivos de audio que serán generados mediante la clase `CreateAudioFactory`.

Así mismo, se utiliza el patrón de diseño *Observer* [Gamma et al., 2003], con la finalidad de saber cuándo una petición ya ha sido atendida o no, por parte del micro-servicio y así evitar generar nuevos archivos de audio que contengan nuevas peticiones, sin antes haber recibido respuesta de la petición anterior. Para el uso de este patrón de diseño, se emplea la interfaz `Observer`, la cual contiene como atributo una instancia de la clase `Subject` y un método `update()`, que tiene como función actualizar el estado del sujeto que está

siendo observado.

Finalmente, la clase `CreateAudioFileFactory` tiene una relación de asociación con la clase `Log`, que permite a la aplicación Android generar una bitácora en donde se registren los fallos que se presenten durante su ejecución. La clase `Log` utiliza el patrón de diseño *Singleton* [Gamma, 2002], el cual permite generar una única instancia de la clase que implemente este patrón, por lo que tiene un constructor de tipo privado. Dentro de los métodos con los que cuenta esta clase se encuentran `getInstance()`, `addLog()` y `getLog()`.

3.1.4.3. Creación y envío de peticiones

Para la creación de peticiones, se utiliza el patrón de diseño *Factory* [Gamma, 2002], que ayuda a crear peticiones de registro de nuevos usuarios y peticiones de usuarios ya registrados. Para el uso de este patrón de diseño, se emplean tres clases que son: `NewUserRequest`, `UserRequest` y `CreateRequestFactory`. Las dos primeras clases hacen uso de la interfaz `Request`, que cuenta con los métodos `prepareRequest()`, `makeRequest()` y `getResponse()`, que son necesarios para poder generar una petición. La tercera clase tiene una relación de asociación con las dos primeras clases mencionadas, con el fin de generar nuevas instancias de dichas clases.

Una vez que ya se tiene implementado el patrón de diseño *Factory*, se utiliza la clase `PostRequest`, cuyo objetivo principal es la generación de un nuevo hilo dentro de la aplicación para poder crear y enviar una petición, sin la necesidad de que la aplicación espere la respuesta del micro-servicio (i.e., que no pueda ejecutarse alguna otra acción durante la espera). Esta clase tiene dos relaciones de agregación, una con la clase `Subject`, que es parte del patrón de diseño *Observer* para el monitoreo del estado de la petición, y otra con la clase `CreateRequestFactory`, que genera la instancia de la petición a enviar. Así mismo, tiene una relación de asociación con la clase `Log` para el registro de las fallas que lleguen a suceder durante la generación o el envío de la petición y otra con la clase `MainActivity`, que es la clase principal de la aplicación.

3.1.4.4. Despliegue y lectura de mensajes en la aplicación

Para poder desplegar mensajes, como instrucciones a seguir o respuestas de peticiones por parte del micro-servicio, se crea la clase `Text`, la cual cuenta con los métodos `showMessage()`, `readMessage()`, `makeBigger()`, `makeSmaller()` y a su vez tiene una relación de composición con la clase `Reader`, encargada de la lectura de los mensajes desplegados mediante voz sintética; esta clase cuenta con un constructor y los métodos `init()`, `initQueue()` y `shutDown()`, y también tiene una relación de asociación con la clase principal `MainActivity`.

Finalmente, la clase `Text` forma parte de los observadores del patrón de diseño *Observer*, por lo que utiliza la interfaz `Observer` para el monitoreo de las respuestas a la petición y el despliegue en pantalla de las mismas.

3.1.4.5. Clase principal

Esta clase es la encargada de la interfaz y el manejo de la aplicación Android, por lo cual cuenta con múltiples métodos como: `onCreate()`, `init()`, `makeRecord()`, `sendRequest()`, `readText()`, `chooseRequest()`, encargados de la implementación adecuada de las clases mencionadas en las sub-secciones anteriores, ya que cuenta con relaciones de asociación con las clases `CreateAudioFileFactory`, `Text`, `PostRequest` y `WiFiConnection`. Esta última clase es la encargada de verificar que se tenga encendido el Wi-Fi y que exista una conexión a Internet dentro del entorno donde se pretende utilizar este sistema, por lo que cuenta con un constructor y los métodos `makeConnection()` y `verifyConnection()`. Así mismo, esta clase tiene una relación de asociación con la clase `Log` para poder registrar las fallas de conexión que se presenten durante la ejecución de la aplicación.

3.2. Sistema de captura permanente

A continuación, se describen los elementos para el diseño del sistema de captura permanente. En la Sección 3.2.1, se describe el funcionamiento del sistema. A continuación, en la Sección 3.2.2, se describen los requerimientos funcionales y no funcionales. Finalmente, en la Sección 3.2.4, se explica la propuesta de solución para el diseño de este sistema.

3.2.1. Funcionamiento del sistema

El objetivo del sistema de captura permanente es tomar las peticiones realizadas por el usuario, mediante su voz, en un entorno más amplio. A diferencia de la aplicación Android, este sistema está diseñado para la captura continua de audio, lo que significa que estos dispositivos de captura permanente siempre están grabando el sonido del ambiente en donde están localizados.

La idea principal de la utilización de estos dispositivos es que el usuario sea capaz de colocar varios de ellos en diferentes ubicaciones dentro de su área de trabajo, ya sea en un sector industrial o en el hogar (e.g., un dispositivo por cuarto). De esta manera, el usuario puede realizar peticiones al micro-servicio, mediante los dispositivos del sistema de captura permanente.

Estas peticiones serán procesadas para obtener la información requerida por el sistema externo con el cual se desee interactuar (e.g., *Broker* o algún dispositivo del Internet de las Cosas, sin la necesidad de mantener un rango de distancia con él, como en el caso de *Alexa* y *Google Assistant*). De esta manera, el sistema de captura permanente, junto con la aplicación Android, pueden capturar en cualquier momento alguna petición del usuario.

Para iniciar una petición, es necesario que el usuario mencione primero la palabra de activación, que indica al sistema la intención de hacer una solicitud. Además de la captura constante de audio, para la detección de la palabra de activación, el sistema cuenta con un umbral de activación que le indica la posibilidad de que un usuario se encuentre hablando, cada vez que este umbral se sobrepasa. Cuando el sistema detecta esta posibilidad, comienza a generar *buffers* con la capacidad de almacenar un segundo de la información de audio. Estos *buffers* son enviados al micro-servicio, mediante el protocolo de comunicación HTTP, para su procesamiento y para identificar si se ha mencionado la palabra de activación, mediante el módulo de procesamiento de lenguaje natural. Si el micro-servicio detecta la palabra de activación, este enviará al sistema de captura permanente una respuesta positiva; en caso contrario, enviará una respuesta negativa, que le permitirá al sistema de captura permanente identificar si debe capturar alguna petición del usuario o permanecer en el modo de escucha continua.

Cuando cualquiera de los dispositivos recibe una respuesta positiva por parte del micro-servicio para capturar una petición de usuario, un LED indicador se enciende, señalando al usuario que puede comenzar a hacer su solicitud. Este LED indicador permanece encendido durante un período de tiempo predeterminado, indicando al usuario que la solicitud se está grabando. Este período de tiempo es lo suficientemente largo como para capturar las peticiones que el micro-servicio puede manejar.

Una vez que el usuario ha completado su solicitud y ha transcurrido el período de tiempo designado, el LED indicador se apaga, indicando al usuario que su solicitud ha sido capturada y enviada al micro-servicio para su procesamiento. Finalmente, después de cumplir con la solicitud del usuario, el dispositivo vuelve a un estado de escucha continua, esperando que el umbral de activación se vuelva a sobrepasar y se repita el procedimiento mencionado.

3.2.2. Requerimientos

A continuación, se presentan los requerimientos funcionales y no funcionales del sistema de captura permanente.

3.2.2.1. Requerimientos funcionales

1. El sistema debe detectar una palabra de activación para iniciar una petición de usuario.
2. El sistema debe generar *buffers* de audio de un segundo de duración y enviarlos al micro-servicio para su procesamiento.
3. Cuando el sistema detecte la palabra de activación, este debe ser capaz de encender un LED que indique al usuario que puede realizar su petición.
4. El sistema debe capturar las solicitudes realizadas por el usuario mediante su voz en tiempo real.
5. El LED debe permanecer encendido durante un período de tiempo predeterminado, mientras se captura la petición del usuario.
6. El sistema apaga el LED una vez que la solicitud ha sido completada y enviada.
7. Las peticiones del usuario deben ser enviadas por el sistema a un micro-servicio para su procesamiento.
8. Una vez capturada y enviada la petición del usuario, el sistema debe regresar a un estado de escucha continua.

3.2.2.2. Requerimientos no funcionales

1. El sistema contará con un archivo de inicialización que contenga los parámetros necesarios para inicializar el sistema.
2. El sistema debe permitir la configuración de un umbral de activación.
3. El sistema debe ser capaz de capturar audio de manera continua para detectar si se ha sobrepasado el umbral de activación.
4. El sistema debe optimizar el uso de recursos, como el manejo adecuado de memoria, para garantizar la correcta generación de *buffers* de almacenamiento de audio.
5. El sistema debe tener un alto grado de precisión en el reconocimiento de la palabra de activación.
6. El período en el que el LED permanece encendido debe ser suficientemente largo para capturar las peticiones del usuario.
7. El sistema debe ser escalable y permitir que múltiples dispositivos se coloquen en diferentes ubicaciones para atender las necesidades del usuario.

3.2.3. Casos de uso

A continuación, se abordan los temas relacionados con los casos de uso del sistema de captura permanente. En primer lugar, se mencionan los actores involucrados, así como sus actividades. Enseguida, se detallan las especificaciones de cada caso de uso.

3.2.3.1. Actores y sus actividades

- Usuario
 - Generar una petición.
- Sistema de captura permanente:
 - Capturar el audio.
 - Detectar la palabra de activación.
 - Encender y apagar el LED indicador.
 - Enviar peticiones de usuario al micro-servicio
- Micro-servicio:
 - Recibir peticiones del sistema de captura permanente.
 - Procesar la petición del usuario.

3.2.3.2. Especificaciones de casos de uso

En la Figura 3.3, se muestra el diagrama de casos de uso referente al sistema de captura permanente para la generación y el envío de peticiones de usuario al micro-servicio.

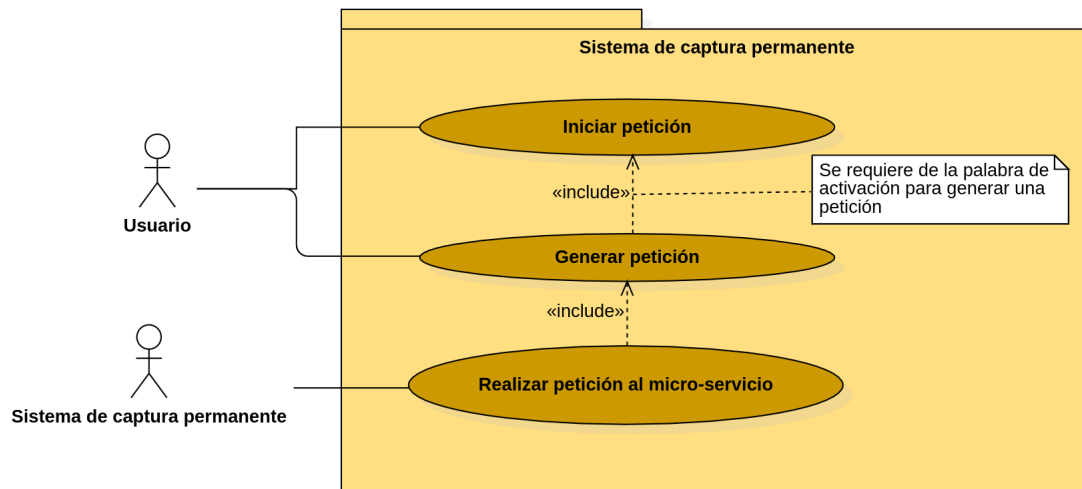


Figura 3.3: Diagrama de casos de uso del sistema de captura permanente

A continuación, se especifica cada uno de los casos de uso del sistema de captura permanente, indicando el actor principal, el objetivo del caso de uso, las precondiciones, el disparador, un escenario, la prioridad, la frecuencia de uso, las excepciones y los actores secundarios.

Caso de uso	Iniciar petición
Actor principal	Usuario
Objetivo	Iniciar una petición de usuario pronunciando una palabra de activación.
Precondiciones	El sistema está conectado a una red Wi-Fi
Disparador	Un usuario desea realizar una petición al micro-servicio

Escenario	<ol style="list-style-type: none"> 1. El usuario menciona la palabra de activación. 2. El sistema detecta que se ha sobrepasado el umbral de activación y genera <i>buffers</i> de audio de un segundo de duración. 3. El sistema lleva a cabo el caso de uso Realizar una petición al micro-servicio 4. El sistema recibe una respuesta positiva del micro-servicio. 5. El sistema enciende el LED indicador. 6. El usuario comienza a realizar su petición.
Flujos alternativos	4.1(a) El sistema recibe una respuesta negativa del micro-servicio, por consiguiente no se comienza con la captura de la petición del usuario y el LED se mantiene apagado.
Prioridad	Alta
Frecuencia de uso	Alta
Excepciones	<ul style="list-style-type: none"> * Si no se tiene una conexión Wi-Fi, el sistema encenderá el LED indicador en amarillo. * Si no se tiene una conexión con el micro-servicio, el sistema encenderá el LED indicador en azul claro con un parpadeo constante de un segundo.
Actores secundarios	Micro-servicio

Caso de uso	Generar una petición de usuario
Actor principal	Usuario
Objetivo	Generar una petición realizada por el usuario mediante voz.
Precondiciones	<ul style="list-style-type: none"> ▪ El sistema identificó la palabra de activación.

Disparador	Un usuario desea realizar una petición al micro-servicio
Escenario	<ol style="list-style-type: none"> 1. El usuario externa su petición. 2. El sistema captura la petición del usuario. 3. El sistema apaga el LED indicador. 4. El sistema lleva a cabo el caso de uso Realizar una petición al micro-servicio. 5. El sistema vuelve al estado de escucha continua.
Flujos alternativos	2.1(a) El sistema no ha podido capturar la petición del usuario, por lo cual se regresa al estado de escucha continua.
Prioridad	Alta
Frecuencia de uso	Alta
Excepciones	* Si se detecta alguna falla de funcionamiento del micrófono encargado de la captura de audio, el sistema enciende el LED indicador en color naranja.
Actores secundarios	Micro-servicio

Caso de uso	Realizar una petición al micro-servicio
Actor principal	Sistema de captura permanente
Objetivo	Enviar una petición mediante <i>buffers</i> de audio y recibir una respuesta del micro-servicio.
Precondiciones	<ul style="list-style-type: none"> ■ Se ha generado un <i>buffer</i> de audio.
Disparador	El usuario requiere realizar una petición al micro-servicio.

Escenario	<ol style="list-style-type: none"> 1. El tiempo establecido para externar la petición ha transcurrido. 2. El sistema verifica que exista una conexión con el micro-servicio, mediante el protocolo HTTP. 3. El sistema envía el <i>buffer</i> de audio al micro-servicio. 4. El micro-servicio atiende la petición del usuario. 5. El sistema recibe una respuesta positiva del micro-servicio. 6. El sistema finaliza la conexión con el micro-servicio.
Flujos alternativos	5.1(a) El sistema recibe una respuesta negativa del micro-servicio y enciende el LED indicador en rojo para después volver al modo de escucha continua.
Prioridad	Alta
Frecuencia de uso	Alta
Excepciones	<ul style="list-style-type: none"> * Si no se tiene una conexión con el micro-servicio, el sistema encenderá el LED indicador en azul claro con parpadeo constante de un segundo. * Si no se tiene respuesta por parte del micro-servicio en un tiempo definido, el sistema cerrará la conexión con el micro-servicio y encenderá el LED indicador en color rojo.
Actores secundarios	Micro-servicio

3.2.4. Diseño de solución

Con base en los requerimientos funcionales y no funcionales necesarios para llevar a cabo el diseño del sistema de captura permanente, se ha tomado la decisión de abordar un estilo de programación concurrente, ya que es necesario llevar a cabo múltiples acciones de manera simultánea, tales como:

- Captura de audio y encendido de LED indicador

- Captura de audio y envío de peticiones al micro-servicio.
- Captura de audio y comparativa de la señal con el umbral indicador.

La programación concurrente es una buena estrategia para generar una solución, ya que permite la ejecución de procesos o tareas múltiples, mediante la generación de sistemas monoprocesador, en donde el dispositivo a utilizar sólo cuenta con un procesador y los procesos o tareas a realizar alternan el tiempo en que se ejecutan, generando una idea de que los procesos se llevan a cabo de manera simultánea. Este paradigma de programación también permite la creación de sistemas multiprocesador en donde el dispositivo cuenta con más de un procesador y permite llevar a cabo tareas realmente de manera simultánea, siempre y cuando el número de procesos o tareas sea igual al número de procesadores [Palma et al., 2003].

Como se puede observar en el diagrama general de la arquitectura del sistema de captura permanente, mostrado en la Figura 3.4, este sistema está conformado por: 1) un módulo de inicialización, 2) cuatro tareas que se ejecutan de manera concurrente 3) una clase `Buffer`, 4) una clase `Petición` y 5) una cola de almacenamiento de tipo *FIFO* (primero en entrar, primero en salir, por sus siglas en inglés), que en conjunto permiten el funcionamiento del sistema de manera adecuada.

El módulo de `Inicialización` es el encargado de realizar la conexión Wi-Fi del sistema para que se pueda llevar a cabo una comunicación cliente-servidor con el micro-servicio. Así mismo, este módulo inicializa el micrófono con el que se pretende realizar la captura de la palabra de activación y la petición del usuario. De igual manera, dicho modulo inicializa la `Tarea 1` y la `Tarea 4`, en donde la primera tarea se encarga de la identificación de la palabra de activación. Si llega a existir algún inconveniente con la conexión Wi-Fi o con la inicialización del micrófono, una bandera localizada en la `Tarea 4` cambia de estado, señalándole al sistema que el LED indicador se debe encender de un determinado color, dependiendo del problema. Con el propósito de evitar el *hardcoding*, se hace uso del `Archivo de configuración`, el cual contiene todos los requerimientos necesarios para la inicialización del sistema como: nombre de la red *Wi-Fi*, contraseña, variables de inicialización del micrófono, entre otros.

Existen dos clases dentro de este sistema: la clase `Buffer`, que permite la generación de *buffers* encargados del almacenamiento de la información de audio, y la clase `Peticiones`, que es utilizada por la `Tarea 2` para el envío de la información de audio almacenada en la `cola de peticiones`. Dentro de la clase `Buffer` se encuentran los métodos necesarios para la generación y manejo de los *buffers* necesarios para el envío de la información al micro-servicio. Los métodos son: `crearBuffer`, `llenarBuffer` y `eliminarBuffer`, los cuales son utilizados por la `Tarea 1`, `Tarea 3` y la `Cola de peticiones`, respectivamente. Por otra parte, la clase `Petición` es la encargada de entablar la comunicación con

el micro-servicio, mediante el protocolo de comunicación HTTP , por lo cual tiene los métodos `EnviarPetición` y `ObtenerRespuesta`, las cuales son utilizados por la Tarea 2 y la Cola de peticiones, respectivamente.

La Tarea 1 está encargada de la identificación de la palabra de activación y hace uso de un semáforo para detener la Tarea 3, que es responsable de la captura de la petición del usuario, y así evitar errores en el sistema. Así mismo, contiene una instancia de la clase `Buffer` para generar los *buffers* de audio de un segundo de duración, utilizando los métodos `CrearBuffer` y `LlenarBuffer`, y enviarlos a la Cola de peticiones para posteriormente ser enviados al micro-servicio por la Tarea 2.

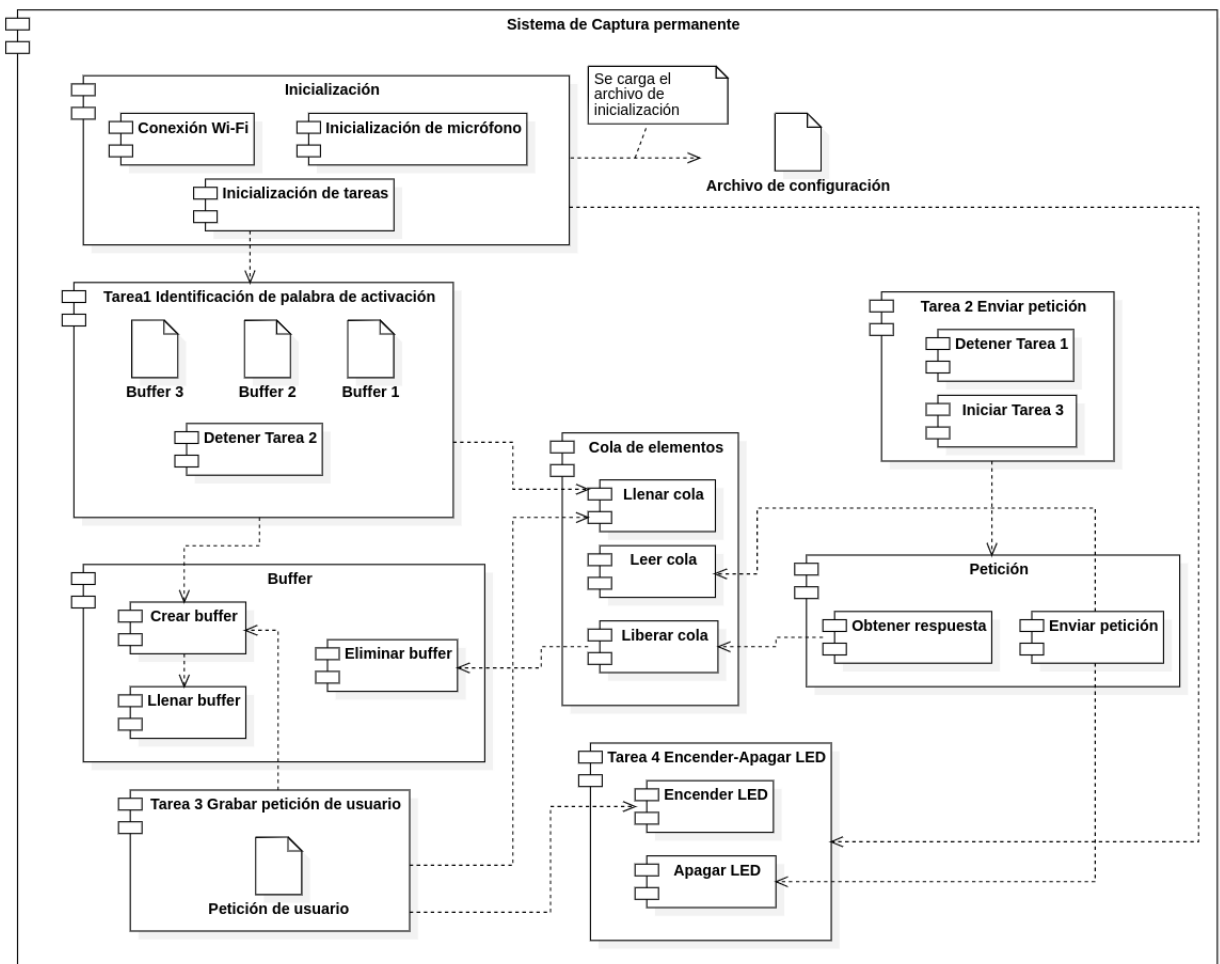


Figura 3.4: Diagrama general de la arquitectura del sistema de captura permanente

La Tarea 2, como se mencionó anteriormente, contiene una instancia de la clase `Petición` para poder entablar una comunicación cliente-servidor con el micro-servicio. Esta tarea

detecta cuando la Cola de peticiones contiene elementos que deben ser enviados. Una vez que se ha detectado que existen elementos dentro de la Cola de peticiones, la Tarea 2 los enviará de uno en uno al micro-servicio, mediante el protocolo de comunicación HTTP. Cada vez que Tarea 2 envía un *buffer* de audio al micro-servicio, este es procesado por el módulo de procesamiento de lenguaje natural del micro-servicio para traducirlo a un formato de texto y detectar si este *buffer* contiene la palabra de activación. Si la palabra de activación ha sido detectada, el micro-servicio envía una respuesta positiva al sistema de captura permanente, indicándole que es momento de dejar de enviar los *buffers* localizados dentro de la Cola de peticiones y eliminarlos para comenzar a grabar la petición del usuario y encender el LED indicador. En caso contrario, si no se ha identificado la palabra de activación, el micro-servicio enviará una respuesta negativa al sistema de captura permanente, indicándole que debe enviar el siguiente *buffer* contenido en la Cola de peticiones, y, al mismo tiempo seguir generando más *buffers* de un segundo, mediante las tareas correspondientes.

Cuando el sistema de captura permanente recibe la respuesta positiva por parte del micro-servicio, indicando que es momento de dejar de generar *buffers* de un segundo y comenzar con la grabación de la petición del usuario, dicho sistema detiene la Tarea 1, encargada de la generación de los *buffers* de un segundo, y activa la Tarea 3 encargada de la generación del *buffer* de petición.

La Tarea 3, al igual que la Tarea 1, cuenta con una instancia de la clase `Buffer`, ya que funciona de manera similar. El objetivo de esta tarea es generar solamente un *buffer* de audio que contenga la cantidad de tiempo establecida para la captura de la petición del usuario. Al mismo tiempo que se activa esta tarea, una bandera localizada en la Tarea 4 cambia de estado, señalándole al sistema que encienda el LED indicador para notificarle al usuario que se ha comenzado a grabar su petición. Esta tarea sólo tiene el propósito de encender el LED indicador de un determinado color, dependiendo de la situación, y apagarlo. Una vez que se ha enviado la petición a la Cola de peticiones, la bandera localizada en la Tarea 4 vuelve a cambiar de estado y el LED indicador se apaga. Finalmente, una vez que se ha enviado la petición al micro-servicio, se detiene la Tarea 3 y se inicia la Tarea 1 para regresar el sistema al modo de escucha continua.

3.3. Micro-servicio para el reconocimiento de voz y habla

A continuación, se realiza una descripción del diseño del micro-servicio para el reconocimiento de voz y habla. En la Sección 3.3.1 se describe el funcionamiento del micro-servicio, seguido de la explicación de los módulos que lo conforman. En la Sección 3.3.2 se aborda el módulo de identificación de usuarios y en la Sección 3.3.3 se explica el módulo de pro-

cesamiento de lenguaje natural. Finalmente, en la Sección 3.3.4 se describe el módulo de comunicación.

3.3.1. Descripción del micro-servicio

Se diseñó un servicio Web bajo la arquitectura de micro-servicios, el cual pretende demostrar la versatilidad del sistema para adaptarse a diferentes ámbitos sin necesidad de estar relacionados. Por esta razón, dentro de sus capacidades, se tienen la comunicación bidireccional con tres tipos de sistemas, los cuales son: 1) micro-servicios que funcionen como intermediarios (*broker*) para delegar las tareas a otros micro-servicios, 2) dispositivos relacionados con el Internet de las Cosas (e.g., persianas automatizadas, brazos robot y lámparas RGB) y 3) sistemas que requieran la traducción de audio capturado a un formato de texto para su posterior procesamiento. Para la realización de esta comunicación se hace uso de una forma estandarizada de describir la interfaz de los dispositivos conocida como *Things Description* [WoT Working Group, 2023]. Así mismo, el sistema tiene la capacidad de atender las peticiones del usuario en idioma español o inglés, dependiendo del ámbito de aplicación.

El micro-servicio, como se observa en la Figura 3.5, cuenta con un módulo de inicialización que permite cargar todas las funcionalidades necesarias, dependiendo si se desea utilizar en idioma Inglés o Español. Por esta razón, al ejecutarse, se selecciona el idioma a utilizar.

Así mismo, cuenta con un módulo de reconocimiento de voz que puede ser habilitado o deshabilitado en la inicialización, dependiendo del entorno y escenario donde se utilice el sistema. Este módulo de reconocimiento de voz brinda al micro-servicio un mayor control de quién puede o no utilizarlo, evitando así en escenarios controlados (e.g., como el sector de la salud) que cualquier persona pueda realizar algún movimiento sin un registro previo. Este registro es realizado haciendo uso de las muestras de audio enviadas por el usuario mediante la aplicación Android.

Independientemente de que el micro-servicio haya utilizado o no el módulo de reconocimiento de voz para la validación de la petición realizada, el micro-servicio convierte el archivo de audio, que contiene la petición del usuario, a un formato de texto y obtiene la información requerida para la interacción con los tres sistemas externos: el *Broker* de un compañero digital, dispositivos del Internet de las Cosas y sistemas que requieran de la traducción de audio a formato de texto. Una vez que se ha obtenido la información relevante para llevar a cabo la petición del usuario, el módulo de procesamiento de lenguaje natural genera un archivo *json* basado en *Things Description* para el envío de la información a los sistemas externos, utilizando el módulo de comunicación.

Si el micro-servicio no ha sido capaz de obtener la información requerida para la generación del archivo en formato *json*, el proceso de la petición se interrumpe y el módulo de comunicación envía al usuario un mensaje que le indica que la petición no ha sido realizada de manera apropiada y que tendrá que volver a hacerla. Del mismo modo, si existe algún inconveniente con la comunicación entre el sistema de reconocimiento de voz y habla con los sistemas externos, se le informará al usuario que su petición no pudo ser atendida en ese momento.

Con base en las tareas que el micro-servicio debe llevar a cabo, se planteó el diseño de los siguientes módulos:

1. Módulo de identificación de usuarios.
2. Módulo de procesamiento de lenguaje natural.
3. Módulo de comunicación.

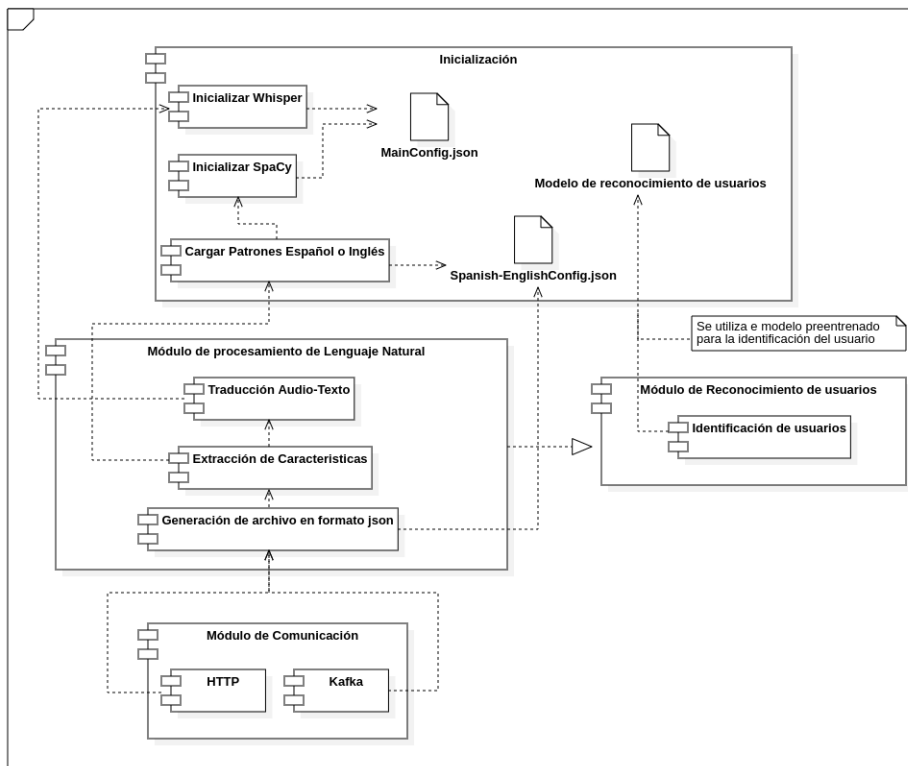


Figura 3.5: Diagrama general de la arquitectura del micro-servicio

3.3.2. Módulo de identificación de usuarios

En esta sección, se presenta el diseño del módulo de identificación de usuarios, comenzando con una explicación de su funcionamiento, seguido de los requerimientos funcionales, requerimientos no funcionales y finalmente los casos de uso.

3.3.2.1. Funcionamiento del módulo

Este módulo está encargado de la identificación de usuarios previamente registrados mediante su voz, que realizan peticiones al micro-servicio, así como de la agregación de nuevos usuarios. El módulo cuenta con un repositorio por cada usuario registrado, en donde se localizan las muestras de audio proporcionadas para su registro. Las muestras de audio son preprocesadas y utilizadas para el entrenamiento de una red convolucional que ayuda a la identificación de los usuarios. El registro de un usuario en el micro-servicio se realiza mediante el uso de un *smartphone*, utilizando la aplicación Android (ver Sección 3.1).

A continuación se menciona el escenario de registro de usuarios:

El usuario presiona el botón colocado en la parte superior derecha de la aplicación Android y selecciona la opción "Agregar nuevo usuario". En respuesta, se despliega un mensaje en el cuadro de texto de la aplicación donde se indica al usuario a registrar que debe presionar el botón de micrófono. Una vez presionado el botón, el usuario debe generar muestras de audio para su registro (con duración de 40 segundos cada una) en donde el usuario se mantenga hablando constantemente. Las muestras son guardadas en el *smartphone* de manera temporal. Para la generación de estas muestras de audio, el usuario sólo debe presionar una vez el botón del micrófono y comenzar a hablar; después de los 40 segundos de grabación, automáticamente se genera un archivo de audio con la muestra y se despliega en el cuadro de texto de la aplicación el mensaje de "muestra (número de muestra) capturada", junto con el mensaje "Presione el botón de micrófono para grabar la siguiente muestra". En caso de ser la última muestra, una vez que se ha guardado en el *smartphone*, se envían al micro-servicio todas las muestras generadas y se despliega el siguiente mensaje en el cuadro de texto de la aplicación: "Muestras enviadas... registrando". Finalmente, se eliminan todos los archivos de audio generados.

Cuando el micro-servicio recibe los archivos de audio, genera un identificador (ID) que está asociado al usuario a registrar y crea un repositorio que lleva por nombre el identificador previamente generado. En este repositorio se guardan las muestras de audio recibidas. Una vez que el micro-servicio ha recibido las muestras para el registro del usuario, estas son segmentadas en audios de un segundo para generar un conjunto de datos que permita el entrenamiento del modelo encargado de la identificación de la voz de usuario. Posteriormente, se procede al pre-procesamiento de las muestras y se realiza el entrenamiento

de la red convolucional utilizada para la generación del modelo.

En caso de que una persona ya se encuentre registrada, esta puede omitir dicho proceso y realizar directamente peticiones mediante las múltiples fuentes de entrada. Una vez que ya se cuenta con un registro y se realiza una petición, el micro-servicio valida al usuario mediante el modelo de aprendizaje automático previamente entrenado y, una vez identificado, se procesa su petición mediante el módulo de lenguaje natural.

3.3.2.2. Requerimientos

A continuación, se listan los requerimientos funcionales y no funcionales del módulo de identificación de personas.

3.3.2.3. Requerimientos funcionales

1. El módulo de identificación de usuarios identificará, mediante voz, a las personas que han realizado una petición y que se encuentren registradas en el micro-servicio.
2. El módulo podrá realizar el registro de voz de nuevos usuarios, mediante la aplicación Android.
3. El micro-servicio generará un repositorio por cada usuario registrado, en donde se tendrán las muestras de audio utilizadas para la generación del modelo de reconocimiento de voz.
4. El micro-servicio re-entrenará el modelo de reconocimiento de voz cada que se agregue un nuevo usuario.
5. El módulo de identificación de usuarios enviará el archivo de audio al módulo de procesamiento de lenguaje natural para su procesamiento, si la petición es aceptada.

3.3.2.4. Requerimientos no funcionales

1. Cada usuario registrado deberá tener un identificador único.
2. Los repositorios generados para cada usuario se localizarán dentro de otro repositorio con el nombre "usuarios".
3. Los repositorios de los usuarios deberán ser llamados por el identificador del usuario.
4. Para el registro de un usuario se requerirán muestras de audio de 40 segundos.
5. Las muestras de audio recibidas para el registro de un usuario serán segmentadas en audios de un segundo de duración.

6. Se desarrollará una red neuronal para la generación del modelo de identificación.
7. El umbral del modelo de identificación (*threshold*) para la identificación de una persona deberá ser igual o superior a 0.8.

3.3.2.5. Casos de uso

A continuación, se abordan los temas relacionados con los casos de uso del módulo de identificación de usuarios. En primer lugar, se mencionan los actores involucrados, así como sus actividades. Enseguida, se detallan las especificaciones de cada caso de uso.

3.3.2.6. Actores y sus actividades

- Módulo de reconocimiento de usuarios:
 - Realizar el registro de nuevos usuarios.
 - Verificar el registro del usuario para aceptar la petición realizada.
- Módulo de procesamiento de lenguaje natural:
 - Recibir archivos de audio con las peticiones del usuario.
- Aplicación Android:
 - Realizar peticiones.
 - Registrar usuarios.
- Sistema de captura permanente:
 - Realizar peticiones.

3.3.2.7. Especificaciones de casos de uso

En la Figura 3.6, se muestra el diagrama de casos de uso referente al módulo de identificación de usuarios.

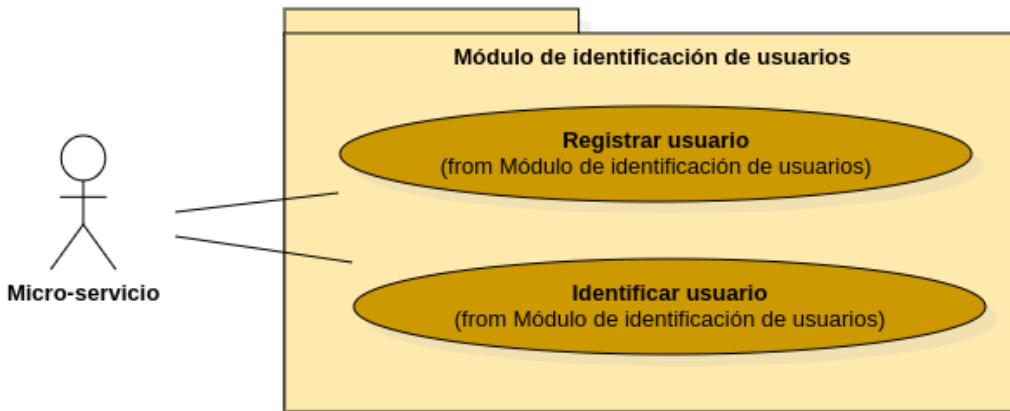


Figura 3.6: Diagrama de casos de uso del módulo de identificación de usuarios

A continuación, se especifica cada uno de los casos de uso del módulo de identificación de usuarios, indicando el actor principal, el objetivo del caso de uso, las precondiciones, el disparador, un escenario, la prioridad, la frecuencia de uso, las excepciones y los actores secundarios.

Caso de uso	Registrar usuario
Actor principal	Módulo de identificación de usuarios
Objetivo	Realizar el registro de un usuario.
Precondiciones	<ul style="list-style-type: none"> El micro-servicio recibe una petición de registro.
Disparador	El micro-servicio recibe una petición de registro por parte de un usuario a través de la aplicación Android

Escenario	<ol style="list-style-type: none">1. La aplicación Android establece una comunicación con el micro-servicio mediante el protocolo de comunicación HTTP.2. El micro-servicio recibe una petición de registro.3. La aplicación Android envía las muestras de audio generadas.4. El módulo de identificación de usuarios genera un identificador para la persona a registrar.5. El módulo de identificación de usuarios genera un repositorio nombrado con el identificador creado y recibe las muestras de audio enviadas por la aplicación Android.6. El módulo de identificación de usuarios segmenta las muestras de audio en archivos de audio de un segundo.7. El módulo de identificación de usuarios guarda las muestras de audio de un segundo en el repositorio creado para la persona a registrar.8. El módulo de identificación de usuarios entrena un nuevo modelo para la identificación de las personas haciendo uso de las muestras de audio de un segundo.9. El módulo de identificación de usuarios actualiza el modelo entrenado para la identificación de personas.10. El micro-servicio envía, como respuesta a la aplicación Android, el mensaje de “Registro finalizado”11. La aplicación Android finaliza la comunicación con el micro-servicio.
-----------	---

Flujos alternativos	<p>Flujo alternativo 1:</p> <p>5.1(a) En el caso de que sea el primer usuario a registrar, se genera el repositorio general donde se guardarán los siguientes repositorios de los usuarios y se continua con el punto 6 del escenario.</p> <p>Flujo alternativo 2:</p> <p>9.1(a) Si el micro-servicio recibe una petición diferente a la de registro durante la generación del nuevo modelo, se lleva a cabo el punto 3 del escenario del caso de uso Identificar a un usuario utilizando el modelo sin actualizar. Una vez realizado el punto 3 de dicho caso de uso, se continua con el punto 10 del escenario de este caso de uso.</p>
Prioridad	Alta
Frecuencia de uso	De baja a media
Excepciones	<p>* Si se pierde la conexión con la aplicación Android, el micro-servicio cancelará la petición de registro y el módulo de identificación de usuarios eliminará los datos almacenados.</p>
Actores secundarios	Aplicación Android

Caso de uso	Identificar usuario
Actor principal	Módulo de identificación de usuarios
Objetivo	Verificar que la petición recibida sea de un usuario registrado.
Precondiciones	<ul style="list-style-type: none"> ■ El micro-servicio recibe una petición que no es de registro.
Disparador	El micro-servicio recibe una petición que no es de registro por parte de un usuario, a través de la aplicación Android o del sistema de captura permanente.

Escenario	<ol style="list-style-type: none"> 1. La Aplicación Android o el sistema de captura permanente establece una comunicación con el micro-servicio. 2. El micro-servicio recibe un archivo de audio que contiene la petición. 3. El módulo de identificación de usuarios identifica a la persona que realizó la petición. 4. El módulo de identificación de usuarios envía el archivo de audio al módulo de procesamiento de lenguaje natural para ser procesado. 5. El módulo de comunicación envía la respuesta de la petición realizada a la aplicación Android o al sistema de captura permanente. 6. La aplicación Android o el sistema de captura permanente finaliza la comunicación con el micro-servicio.
Flujos alternativos	3.1 Si el usuario no ha podido ser identificado se salta al punto 5 del escenario, enviando como respuesta el mensaje de "Usuario no identificado" y se recomienda realizar una petición de registro.
Prioridad	Alta
Frecuencia de uso	Alta
Excepciones	* Si se pierde la conexión con la aplicación Android o el sistema de captura permanente, el micro-servicio cancelará la petición realizada.
Actores secundarios	Módulo de procesamiento de lenguaje natural, aplicación Android y sistema de captura permanente

3.3.3. Módulo de procesamiento de lenguaje natural

Dentro de esta sección, se presenta el diseño del módulo de procesamiento de lenguaje natural, comenzando con una explicación del funcionamiento del módulo, seguido de los requerimientos funcionales, requerimientos no funcionales y finalmente los casos de uso.

3.3.3.1. Funcionamiento del módulo

Este módulo tiene como tareas principales la traducción de archivos de audio a texto y la obtención de la información necesaria para la ejecución de la petición del usuario. Dependiendo de la configuración, este módulo se activa después de la identificación del usuario o directamente al recibir una petición.

El módulo tiene la capacidad de traducir las peticiones realizadas por la aplicación Android o el sistema de captura permanente en el idioma español o inglés a un formato de texto. Una vez realizada la traducción, se hace uso de herramientas de procesamiento de lenguaje natural que permiten utilizarse de manera local, sin la necesidad de utilizar el cómputo en la nube con el fin de garantizar la privacidad de la información del usuario.

El módulo es capaz de identificar el sistema con el que necesita interactuar para ejecutar la petición del usuario. Utiliza técnicas de procesamiento de lenguaje natural para la obtención de la información requerida para la generación de un archivo en formato *json* que se envía al sistema con el que se desea interactuar, mediante el módulo de comunicación, para la interacción con los sistemas que permitan la ejecución de la petición del usuario.

El módulo cuenta con la capacidad de identificar si la petición solicitada es posible de ejecutarse o no, tomando en consideración el idioma en que se ha inicializado el micro-servicio y los sistemas a interactuar (e.g., el micro-servicio sólo puede interactuar con dispositivos IoT si la petición se realiza en inglés). En caso de que la petición no sea viable, se le informa al usuario que la petición realizada no ha podido ser identificada. Así mismo, si el módulo no logra obtener toda la información requerida para la generación del archivo en formato *json*, necesario para la comunicación con algún sistema, se le hace saber al usuario que la petición no fue realizada correctamente.

Este módulo es esencial para garantizar que las solicitudes de los usuarios se ejecuten de manera efectiva y eficiente, proporcionando una experiencia fluida y una comunicación precisa con los sistemas asociados.

3.3.3.2. Requerimientos

A continuación, se en listan los requerimientos funcionales y no funcionales del módulo de procesamiento de lenguaje natural.

3.3.3.3. Requerimientos funcionales

1. El módulo deberá interactuar con el módulo de identificación de usuarios (ver Sección 3.3.2) para la identificación del usuario antes de realizar alguna acción, si se encuentra habilitada esta funcionalidad.
2. El módulo deberá ser capaz de traducir archivos de audio en español e inglés a un formato de texto.
3. El módulo identificará que tipo de peticiones es capaz de ejecutar dependiendo del idioma que se utilice.
4. El módulo debe ser capaz de obtener la información necesaria para la ejecución de la petición del usuario.
5. El micro-servicio debe ser capaz de informar al usuario cuando una solicitud no puede ser identificada o si falta información para generar el archivo en formato *json*.
6. El módulo debe generar un archivo en formato *json* que se envíe a través del módulo de comunicación.

3.3.3.4. Requerimientos no funcionales

1. El módulo debe procesar la información de manera local para garantizar la privacidad de los datos del usuario y evitar el uso de cómputo en la nube.
2. El módulo hará uso del *framework Whisper* para la traducción del formato de audio a texto y la obtención de información relevante.
3. El módulo debe procesar las solicitudes en un tiempo razonable para proporcionar una experiencia de usuario eficiente.

3.3.3.5. Casos de uso

A continuación, se abordan los temas relacionados con los casos de uso del módulo de procesamiento de lenguaje natural. En primer lugar, se mencionan los actores involucrados, así como sus actividades. Después, se detallan las especificaciones de cada caso de uso.

3.3.3.6. Actores y sus actividades

- Módulo de procesamiento de lenguaje natural:
 - Traducir el archivo de audio a un archivo de texto.
 - Extraer información relevante para la ejecución de las peticiones.

- Módulo de comunicación:
 - Recibir la información extraída en un formato *json* para su envío.
 - Enviar la información en formato *json* a sistemas externos para la ejecución de la petición del usuario.

3.3.3.7. Especificaciones de casos de uso

En la Figura 3.7, se muestra el diagrama de casos de uso referente al módulo de procesamiento de lenguaje natural.

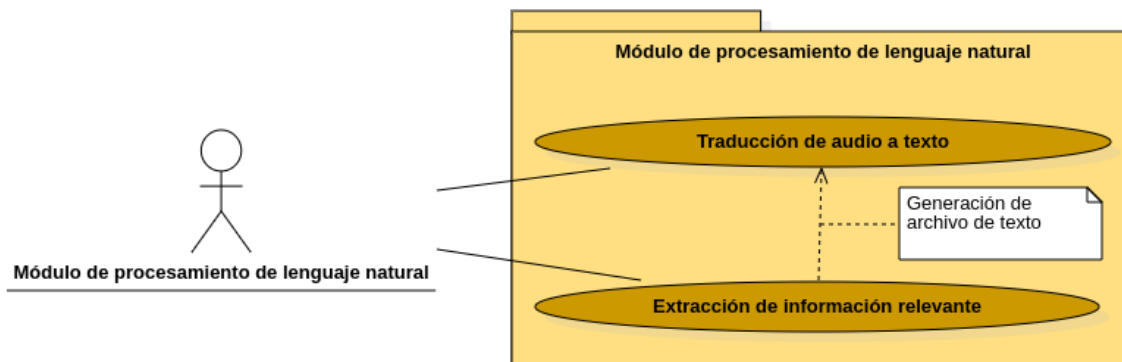


Figura 3.7: Diagrama de casos de uso del módulo de procesamiento de lenguaje natural

A continuación, se especifica cada uno de los casos de uso del módulo de procesamiento de lenguaje natural, indicando el actor principal, el objetivo del caso de uso, las precondiciones, el disparador, un escenario, la prioridad, la frecuencia de uso, las excepciones y los actores secundarios.

Caso de uso	Traducción de audio a texto
Actor principal	Módulo de procesamiento de lenguaje natural
Objetivo	Traducir un archivo de audio en un archivo de texto en los idiomas español o inglés.
Precondiciones	<ul style="list-style-type: none"> ▪ Se ha identificado al usuario que solicitó la petición, si esta opción se encuentra habilitada.
Disparador	El módulo de identificación de usuarios detecta al usuario que hizo la petición.
Escenario	<ol style="list-style-type: none"> 1. El módulo de procesamiento de lenguaje natural recibe el archivo de audio. 2. El módulo de procesamiento de lenguaje natural realiza la traducción del archivo de audio a texto en español o inglés.
Flujos alternativos	<p>1.1(a) Si la opción de identificación de usuarios está habilitada y el usuario no ha sido identificado, se enviará un mensaje a la aplicación Android sugiriéndole una petición de registro.</p> <p>2.1(a) Se ha realizado una petición en un idioma diferente al español o inglés, por lo cual se envía un mensaje al usuario indicándole que el idioma utilizado no es válido.</p>
Prioridad	Alta
Frecuencia de uso	Alta
Excepciones	<p>* Si el archivo de audio proporcionado no cumple con los requisitos de formato, el proceso de traducción no puede llevarse a cabo y se le notifica al usuario sobre la invalidez del formato.</p>
Actores secundarios	No

Caso de uso	Extracción de información relevante
Actor principal	Módulo de procesamiento de lenguaje natural
Objetivo	Extraer información relevante de las transcripciones de texto para la ejecución de las solicitudes del usuario.
Precondiciones	<ul style="list-style-type: none"> ■ Se ha transcrito la petición del usuario.
Disparador	Se ha transcrito la petición del usuario.
Escenario	<ol style="list-style-type: none"> 1. El módulo de procesamiento de lenguaje natural ha transcrito el archivo de audio a texto. 2. El módulo de procesamiento de lenguaje natural procesa la transcripción de texto para identificar el sistema externo con el que debe iniciar una comunicación. 3. El módulo de procesamiento de lenguaje natural extrae la información necesaria. 4. El módulo de procesamiento de lenguaje natural organiza la información en formato <i>json</i>. 5. El módulo de procesamiento de lenguaje natural envía al módulo de comunicación el archivo en formato <i>json</i> generado.
Flujos alternativos	<p>2.1(a) No se ha identificado algún sistema externo con el micro-servicio pueda interactuar, por lo cual se envía un mensaje al usuario notificándole que su petición no es válida.</p> <p>3.1(a) Si no se puede generar de manera apropiada el archivo en formato <i>json</i>, se le notifica al usuario que su petición ha sido incompleta y se pide realizarla de nuevo.</p>
Prioridad	Alta
Frecuencia de uso	Alta

Excepciones	* Si se pierde la conexión con la aplicación Android o el sistema de captura permanente, el micro-servicio cancelará la petición realizada.
Actores secundarios	Módulo de comunicación

3.3.4. Módulo de comunicación

En esta sección se describe el funcionamiento del módulo de comunicación, así como sus requerimientos funcionales, no funcionales y casos de uso.

3.3.4.1. Funcionamiento del módulo

El objetivo principal de este módulo es enviar la información extraída y estructurada en formato *json*, proveniente del módulo de procesamiento de lenguaje natural, a sistemas externos que permiten la ejecución de las peticiones de los usuarios.

El módulo es capaz de facilitar la interoperabilidad con los sistemas externos mencionados, tanto para la recepción como para el envío de datos en el formato *json*. Por esta razón, cumple con la flexibilidad para adaptarse a diversas necesidades y requisitos.

Con el propósito de diseñar un sistema versátil, el módulo utiliza el protocolo HTTP para la comunicación con sistemas externos, tanto para la recepción como el envío de la información extraída por el módulo de procesamiento de lenguaje natural. No obstante, este módulo permite la utilización de métodos alternativos de comunicación que le permitan interactuar con sistemas de mensajería y eventos, brindando la flexibilidad necesaria para adaptarse a sistemas externos.

3.3.4.2. Requerimientos

A continuación, se presentan los requerimientos funcionales y no funcionales de este módulo.

3.3.4.3. Requerimientos funcionales

1. El módulo de comunicación debe ser capaz de recibir información estructurada en formato *json*, desde el módulo de procesamiento de lenguaje natural.

2. El módulo debe ser capaz de comunicarse con sistemas externos mediante solicitudes HTTP para la recepción y el envío de datos estructurados.
3. El módulo debe admitir métodos alternativos de comunicación que permitan la interacción con sistemas de mensajería y eventos.

3.3.4.4. Requerimientos no funcionales

1. El módulo de comunicación debe ser capaz de adaptarse a diferentes protocolos de comunicación, como HTTP y otros métodos alternativos, para garantizar la interoperabilidad con sistemas externos variados.
2. El módulo de comunicación debe garantizar un rendimiento óptimo al enviar y recibir datos.

3.3.4.5. Casos de uso

A continuación, se abordan los temas relacionados con los casos de uso del módulo de comunicación. En primer lugar, se mencionan los actores involucrados, así como sus actividades. Enseguida, se detallan las especificaciones de cada caso de uso.

3.3.4.6. Actores y sus actividades

- Módulo de procesamiento de lenguaje natural:
 - Traducir el archivo de audio a un archivo de texto.
 - Extraer información relevante para la ejecución de una petición.
- Módulo de comunicación:
 - Recibir la información extraída de un archivo en un formato *json* para su envío a un sistema externo.
 - Enviar la información en formato *json* a sistemas externos para la ejecución de la petición del usuario.
 - Recibir información de los sistemas externos sobre el estado de la petición del usuario.

3.3.4.7. Especificaciones de casos de uso

En la Figura 3.8, se muestra el diagrama de casos de uso referente al módulo de comunicación.

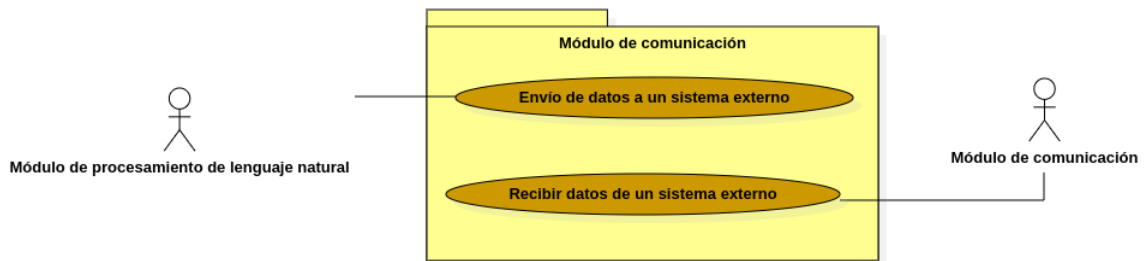


Figura 3.8: Diagrama de casos de uso del módulo de comunicación

A continuación, se especifica cada uno de los casos de uso del módulo de comunicación, indicando el actor principal, el objetivo del caso de uso, las precondiciones, el disparador, un escenario, la prioridad, la frecuencia de uso, las excepciones y los actores secundarios.

Caso de uso	Envío de datos a un sistema externo
Actor principal	Módulo de procesamiento de lenguaje natural
Objetivo	Enviar información estructurada en formato <i>json</i> a un sistema externo.
Precondiciones	<ul style="list-style-type: none"> ▪ Debe existir una conexión de red Wi-Fi.
Disparador	El módulo de procesamiento de lenguaje natural ha generado un archivo en formato <i>json</i> , de manera apropiada.
Escenario	<ol style="list-style-type: none"> 1. El módulo de procesamiento de lenguaje natural envía el archivo en formato <i>json</i> al módulo de comunicación. 2. El módulo de comunicación establece una conexión con el sistema externo a través del protocolo de comunicación indicado. 3. El sistema externo recibe y procesa los datos.
Flujos alternativos	2.1(a) No se logra establecer una conexión con el sistema externo, por lo cual el módulo de comunicación notifica el error de conexión.
Prioridad	Alta

Frecuencia de uso	Alta
Excepciones	* Si se pierde la conexión con el sistema externo, se cancelará la solicitud realizada.
Actores secundarios	Sistema externo

Caso de uso	Recibir datos de un sistema externo
Actor principal	Módulo de comunicación
Objetivo	Recibir información estructurada en formato <i>json</i> de un sistema externo.
Precondiciones	<ul style="list-style-type: none"> ▪ Se ha establecido una conexión con el sistema externo.
Disparador	El sistema externo envía una respuesta a la solicitud realizada por el módulo de comunicación.
Escenario	<ol style="list-style-type: none"> 1. El módulo de comunicación se mantiene en espera de la respuesta del sistema externo. 2. El módulo de comunicación recibe la respuesta del sistema externo y la procesa. 3. El módulo de comunicación informa al usuario que su petición ha sido procesada exitosamente.
Flujos alternativos	2.1(a) No se recibe respuesta por parte del sistema externo después de un tiempo determinado, por lo cual el módulo de comunicación cancela la conexión con el sistema externo y solicita al usuario que vuelva a realizar su petición.
Prioridad	Alta
Frecuencia de uso	Alta

Excepciones	* Si se pierde la conexión con el sistema externo, se cancelará la solicitud realizada.
Actores secundarios	Sistema externo

3.3.5. Clases del micro-servicio

En esta sección se explican las clases que conforman el micro-servicio para poder funcionar de manera adecuada. Así mismo, se mencionan las relaciones que tienen estas clases entre ellas y cuáles son los atributos y métodos con los que cuentan.

3.3.5.1. Clases e interfaces

EL micro-servicio cuenta con un total de 16 clases y tres interfaces, las cuales se pueden observar en la Figura 3.9. Las clases son las siguientes: `MainConfig`, `SpanishConfig`, `EnglishConfig`, `EnglishPatterns`, `SpanishPatterns`, `Repository`, `IoTCommunication`, `AudioFile`, `Language`, `BandFilter`, `App`, `Model`, `FilterFactory`, `BrokerCommunication`, `DataSet` y `RelevantInformation`. Las interfaces definidas son: `IFilter`, `IPattern` e `IDevice`.

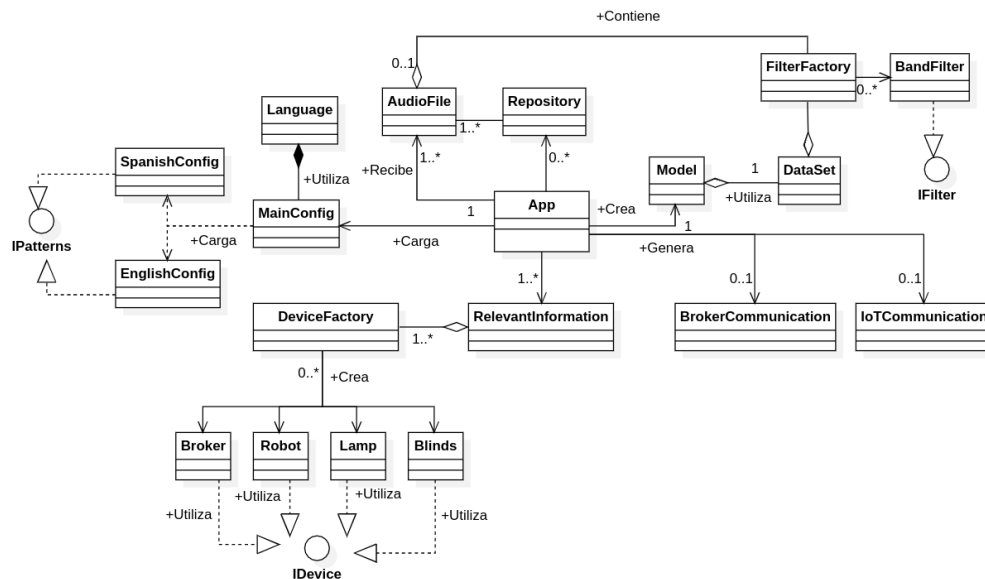


Figura 3.9: Diagrama de clases del micro-servicio para el reconocimiento de voz y habla multi-fuente orientado a compañeros digitales.

3.3.5.2. Inicialización del micro-servicio

Para la configuración inicial del micro-servicio y para evitar la práctica de *hardcoding*, es necesario contar con tres archivos de configuración en formato *json*. El primer archivo, denominado `MainConfig.json`, contiene todos los datos esenciales para la configuración inicial de los módulos del sistema, además de contener los nombres de los archivos para la configuración del idioma que se empleará en el micro-servicio.

Para la configuración del idioma español se utiliza el archivo `SpanishConfig.json` y para el idioma inglés el archivo `EnglishConfig.json`. Ambos archivos contienen la información requerida por las herramientas de procesamiento lenguaje natural para la extracción de información en el idioma correspondiente. Además, los archivos contienen las estructuras de los archivos *json* que son requeridos para la comunicación con los sistemas externos, así como sus direcciones utilizadas en el módulo de comunicación. Finalmente, se encuentran los mensajes para la interacción con usuarios dependiendo de las respuestas obtenidas de los sistemas externos.

Las clases `MainConfig` y `Language` son utilizadas para la configuración inicial del micro-servicio. Ambas clases cuentan con un constructor y tienen una relación de composición. Dentro de los métodos con los que cuenta la clase `Language` se encuentran: `getConfigFile`, `loadConfigFile` y `closeConfigFile`. Algunos de los métodos que conforman la clase `MainConfig` son: `startModules`, `getLanguage`, `readConfigFile` y el método `getConfigFile`.

3.3.5.3. Generación de repositorios

Dentro del micro-servicio, es necesaria la generación de múltiples repositorios para poder almacenar la información requerida (e.g., archivos de audio con muestras de nuevos usuarios y con peticiones). Por consiguiente, se ha generado la clase `Repository`, la cual cuenta con los métodos `createRepository` y `deleteRepository` y los parámetros requeridos por esta clase son: el nombre del repositorio a crear y la dirección donde se generará.

3.3.5.4. Manipulación de los archivos de audio

Para poder guardar y manipular los archivos de audio, se generó la clase `AudioFile`. Esta clase cuenta con tres métodos, `savefile`, `savefiles` y `filesegmentation`, los cuales ayudan a guardar y segmentar las muestras de audio de nuevos usuarios para su posterior uso en el entrenamiento del modelo para la identificación de usuarios y para guardar las peticiones de usuarios ya registrados para su posterior análisis por el módulo de procesamiento de lenguaje natural. La clase tiene una relación de asociación con la clase `App`.

3.3.5.5. Preprocesamiento del conjunto de datos

Para el preprocesamiento de los datos, se generaron diversas clases, dentro de las cuales se encuentran la clase `DataSet` y el conjunto de clases `BandFilter` y `FilterFactory` que, junto con la interfaz `IFilter`, conforman de nuevo el patrón de diseño *Factory* (ver Figura 3.9). Se tomó esta decisión de diseño con la finalidad de poder agregar diferentes filtros para el tratamiento de la señal de audio, según sea necesario, sin la necesidad de tener que modificar de manera significativa el código del micro-servicio.

Las clases que utilizan la interfaz `IFilter`, recurren a su método llamado `createfilter`, el cual dentro de sus atributos define la frecuencia de corte y el orden de los filtros a utilizar. Así mismo, la clase `DataSet` cuenta con múltiples métodos encargados de realizar el procesamiento adecuado para que los datos (archivos de audio) puedan ser utilizados en el modelo de identificación de usuarios. Esta clase cuenta con el método `getclassnames` encargado de obtener el identificador de cada repositorio de usuarios que se ha generado para posteriormente ser utilizado en el método `classifydataset`, que se encarga de identificar los elementos que se encuentran dentro de cada repositorio de usuarios. Esta clase cuenta también con el método `shuffledataset` que, como su nombre lo indica, está encargado de mezclar los datos obtenidos para que, posteriormente, con el método `createdataset` se generen los conjuntos de datos de entrenamiento y de validación, dándoles un porcentaje a cada uno de estos conjuntos de manera que, en total, se cubra el 100% de los datos. Finalmente, la misma clase `DataSet` cuenta con un método llamado `audiotofft`, que se encarga de aplicar la transformada rápida de Fourier (FFT, por sus siglas en inglés - *Fast Fourier Transform*) a cada una de las muestras, tanto del conjunto de prueba como del conjunto de validación, para posteriormente ser utilizadas para generar el modelo de identificación de usuarios.

3.3.5.6. Generación del modelo de identificación de usuarios

Para la generación del modelo de reconocimiento de usuarios se utiliza la clase `Model`, la cual cuenta con los atributos requeridos para el entrenamiento del modelo que se genera por la red convolucional utilizada. Así mismo, cuenta con los métodos `buildModel`, `trainModel` y `loadModel`, que permiten la generación y el manejo del modelo de identificación de usuarios. Para su correcto funcionamiento cuenta con una relación de agregación con la clase `DataSet` la cual es indispensable para la generación del modelo.

3.3.5.7. Obtención de información relevante

El módulo de procesamiento de lenguaje natural está conformado por N clases, las cuales ayudan a la conversión del archivo de audio a texto y a la extracción de las características esenciales para la interacción con los sistemas externos en el idioma español o

inglés.

Para cada idioma se tiene una clase que contiene los elementos necesarios para la extracción de la información relevante. Estas clases son `SpanishConfig` y `EnglishConfig`, las cuales son utilizadas junto con la clase `MainConfig` al momento de la inicialización del micro-servicio. Estas clases utilizan una interfaz denominada `IPatterns` la cual contiene dos métodos a utilizar, `createPatterns` y `chargePatterns`.

Para la identificación del dispositivo con el que se desea interactuar se hace uso de la clase `RelevantInformation`, la cual contiene los métodos `audioToText` y `getIntention` encargados de la traducción del archivo de audio a texto y de la identificación del dispositivo a utilizar, respectivamente, dependiendo del idioma seleccionado. Esta clase a su vez contiene una instancia de la clase `DeviceFactory` para la generación de instancias referentes a los sistemas externos con los que se desea interactuar (e.g., generación de la clase `Robot` y `Lámpara`). Las clases de los dispositivos externos deberán hacer uso de la interfaz `IDevice` que contiene el método `getFeatures`, el cual permite obtener la información para la generación del archivo en formato *json*, requerido para la comunicación con los sistemas externos.

3.3.5.8. Envío de comandos

Una vez que se tienen las características de la petición en el formato *json* adecuado para poder ser enviadas, se utilizan las clases `BrokerCommunication` para el envío de la información al micro-servicio *Broker* y la clase `IoTCommunication` para el envío de la información al dispositivo IoT adecuado. Ambas clases tienen una relación de asociación directa con la clase `App`.

La clase `BrokerCommunication` tiene como atributos la dirección IP del *Broker* al que se le envía la información y tiene como parámetro de entrada la información obtenida por parte del módulo de procesamiento de lenguaje natural. Cuenta con los métodos `producer`, responsable de enviar la información, y `consumer`, encargado de recibir la respuesta del micro-servicio *Broker*.

La clase `IoTCommunication`, al igual que la clase `BrokerCommunication` mencionada anteriormente, contiene como atributos las direcciones IP de los dispositivos con los que intercambia información, así como los métodos encargados de la comunicación mediante el protocolo HTTP (GET, POST, PUT y DELETE) para el envío de información y recepción de las respuestas de los dispositivos IoT.

Capítulo 4

Implementación del sistema de reconocimiento de voz y habla multi-fuente

En este capítulo se aborda la implementación de los elementos que constituyen al sistema de reconocimiento de voz y habla multi-fuente para compañeros digitales. En la Sección 4.1.1 se describe la implementación de la Aplicación Android, seguida de la sección 4.2 en donde se aborda la implementación del sistema de captura permanente. Finalmente, en la Sección 4.3 se describe la implementación del micro-servicio desarrollado.

4.1. Aplicación Android

En esta sección, se aborda la implementación de la aplicación Android, la cual es una de las fuentes de entrada del sistema de reconocimiento de voz y habla multi-fuente. En la Subsección 4.1.1 se describe la interfaz de usuario de la aplicación, seguida de la Subsección 4.1.2 en donde se tratan los aspectos técnicos de la aplicación desarrollada.

4.1.1. Interfaz de usuario

El diseño de esta aplicación tiene como objetivo principal facilitar la interacción entre los usuarios y un micro-servicio encargado del procesamiento de las solicitudes de usuario. Estas solicitudes son enviadas a los sistemas externos para poder interactuar con ellos y llevar a cabo las peticiones de los usuarios.

La aplicación constantemente guía al usuario en las acciones que debe ejecutar para realizar su petición, mediante dos modos: el primero es utilizando mensajes mostrados en un cuadro de texto y el segundo es haciendo uso de voz sintética. La aplicación está diseñada para cumplir con dos tareas específicas: 1) enviar peticiones de los usuarios a

un micro-servicio y 2) notificar al usuario sobre algún evento relevante con respecto a su petición.

Para llevar a cabo la primera tarea, se decidió implementar un botón para la generación y el envío de peticiones, con el objetivo de brindar al usuario la posibilidad de controlar estas acciones y la cantidad de información a enviar en un periodo preestablecido. Este periodo es de 15 segundos, lo cual permite al usuario poder realizar peticiones que conlleven diversas especificaciones. Si el usuario sobrepasa este tiempo, su petición se cancelará y se le informará, mediante un mensaje de texto desplegado en la ventana de la aplicación Android, que el tiempo ha terminado y es necesario realizar de nuevo la petición. Por otro lado, si se ha terminado la petición en un tiempo menor a los 15 segundos, el usuario sólo deberá volver a presionar el botón y su petición es enviada. Como se muestra en la Figura 4.1, este botón se localiza en la parte inferior central de la ventana de la aplicación Android y se muestra mediante un icono de micrófono disponible en el entorno de desarrollo Android Studio. Este icono sirve como indicador para el usuario de la acción que realiza.

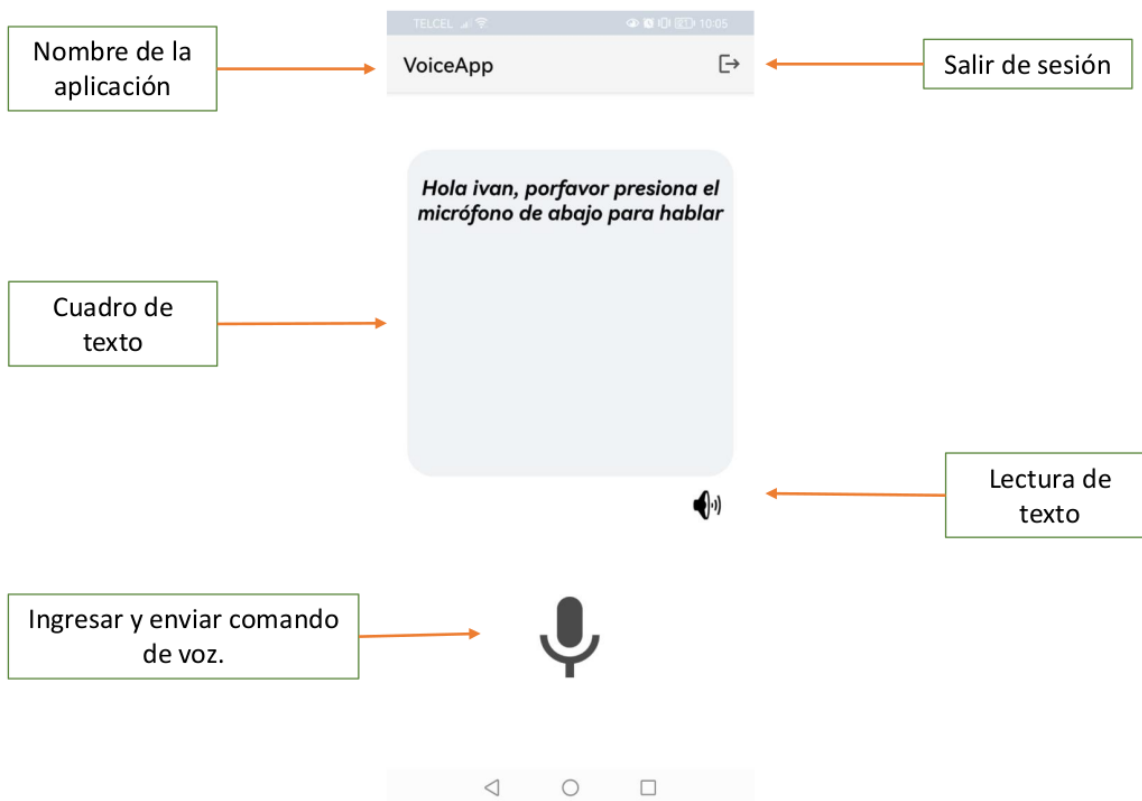


Figura 4.1: Componentes de la primera versión de la aplicación para la captura y envío de peticiones.

Para realizar la segunda tarea, se decidió utilizar un cuadro de texto localizado en la parte central, abarcando la mayor parte de ventana, y un segundo botón de menor tamaño, colocado en la parte inferior derecha del cuadro de texto, con el fin de indicarle al usuario que este botón con el icono de bocina está relacionado con el cuadro de texto. Finalmente, se colocó un tercer botón localizado en la parte superior derecha de la ventana con un icono designando la salida de la aplicación.

Esta primera versión de la aplicación cubre la mayor parte de las necesidades de los usuarios al utilizarla, ya que permite el envío de sus peticiones y brinda un medio para la interacción entre el usuario y el micro-servicio, utilizando mensajes de texto o voz sintética. Las dos maneras, en las que el usuario puede obtener información, están enfocadas en atender a personas que puedan sufrir de algún problema relacionado con la visión y les sea complicado leer los mensajes de texto. Sin embargo, al realizar algunas pruebas con personas mayores, se pudo notar existe la posibilidad de que padezcan tanto de una deficiencia en su visión como en su capacidad para escuchar, especialmente si sufren de alguna enfermedad como la diabetes. Por lo cual, la interfaz de usuario de esta primera versión no estaba totalmente adecuada para estos casos.

Se realizó una segunda interfaz mostrada en la Figura 4.2, realizando dos modificaciones a la primera versión. La primera modificación fue agregar una nueva funcionalidad a la aplicación que permita manipular el tamaño de fuente de los mensajes de texto mostrados al usuario. Esta funcionalidad se agregó utilizando dos nuevos botones a la interfaz, utilizando los iconos de mas (+) y menos (-), del entorno de desarrollo de Android Studio. Estos dos botones, en un principio, fueron colocados a un lado del botón con el icono de bocina. Sin embargo, al momento de realizar algunas pruebas, se notó que los usuarios asociaban esos botones como los encargados de controlar el volumen de la voz sintética. Finalmente, los botones fueron reubicados en la parte inferior central del cuadro de texto, dando mayor espacio entre estos dos botones y el botón encargado de la voz sintética.

Así mismo, como se observa en la Figura 4.2, se cambio el botón de la parte superior derecha por un menú, donde el usuario puede seleccionar entre cuatro opciones. La primera opción hace referencia a comenzar el proceso de registro para un nuevo usuario en el micro-servicio, la segunda y tercera opción hacen referencia al tipo de formato en que se desea enviar la petición al micro-servicio y la cuarta opción permite salir de la aplicación.

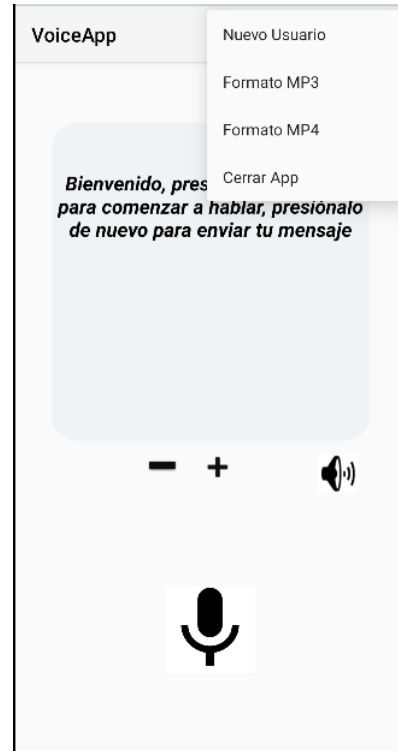
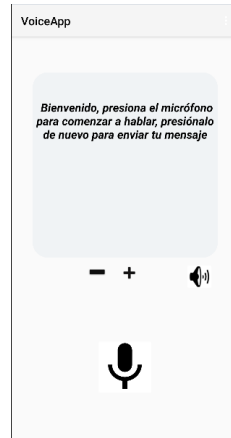


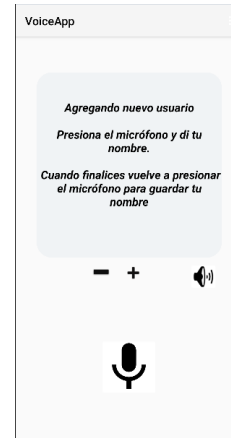
Figura 4.2: Componentes de la versión final de la aplicación para la captura y envío de peticiones.

Como se mencionó al inicio de esta sección y como se puede observar en las Figuras 4.3a y 4.3b, la aplicación es interactiva, ya que guía al usuario mediante los mensajes de texto o por medio de la voz sintética sobre las acciones que debe realizar para hacer una petición al micro-servicio o para comenzar un proceso de registro de usuario en el micro-servicio. Así mismo, se muestran al usuario diferentes tipos de mensajes referentes al proceso de su petición. Entre estos se encuentran mensajes referentes a:

1. Peticiones fallidas debido a una mala conexión
2. Peticiones no reconocidas
3. Peticiones incompletas
4. Peticiones exitosas



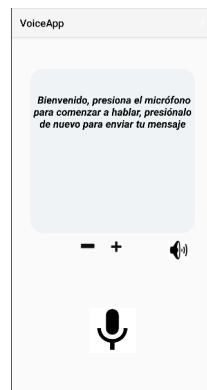
(a) Mensaje para generar una petición



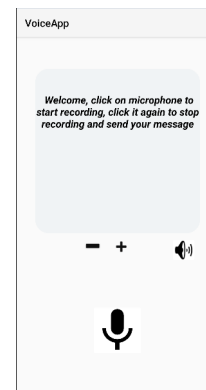
(b) Mensaje para registrar un nuevo usuario

Figura 4.3: Mensajes de interacción para guiar al usuario (1/3).

La interfaz de usuario también tiene la funcionalidad de adecuarse a dos tipos de idiomas, como se muestra en las Figuras 4.4a y 4.4b. El primero de ellos es el idioma español y el segundo es el idioma inglés. Dependiendo de la configuración de la aplicación, los mensajes son mostrados al usuario en el idioma seleccionado; así mismo, la voz sintética utilizada cambia de idioma para poder realizar la lectura de los mensajes mostrados. Para poder habilitar esta funcionalidad, se realizaron dos archivos de configuración denominados `string-es` y `string-en`, referentes a los idiomas español e inglés, respectivamente. Estos archivos contienen toda la información relevante para el correcto funcionamiento de la aplicación Android. Dentro de los parámetros que son parte de estos archivos, se encuentran: `endpoints` utilizados para la comunicación con el micro-servicio, mensajes para guiar al usuario en la realización de diferentes acciones, nombre de los archivos generados que contienen la petición del usuario, etc.



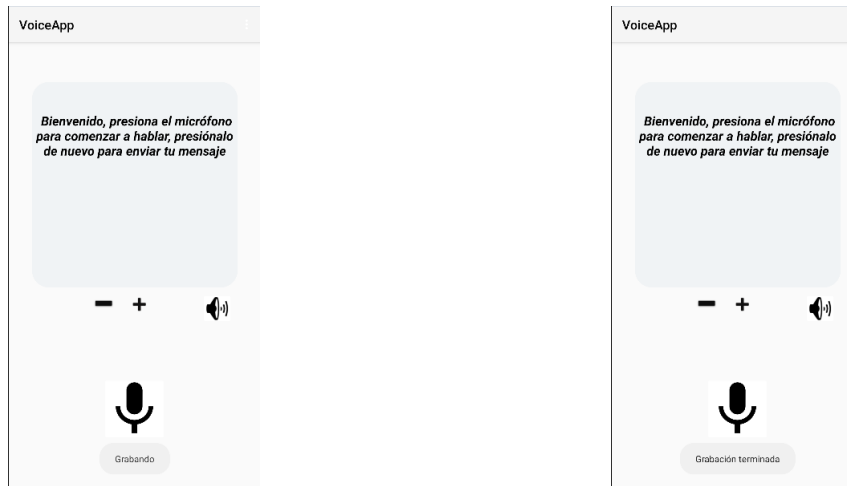
(a) Mensaje en el idioma español.



(b) Mensaje en el idioma inglés.

Figura 4.4: Mensajes de interacción para guiar al usuario (2/3).

Finalmente, la aplicación también cuenta con mensajes de corta y larga duración, los cuales son mostrados cuando un usuario comienza a grabar su petición, al momento de tocar por primera vez el botón de micrófono, y cuando se termina la grabación y se envía el mensaje, al tocar por segunda vez el botón de micrófono, como se puede observar en las Figuras 4.5a y 4.5b. Esos mensajes son implementados para hacerle saber al usuario las acciones que se están llevando a cabo.



(a) Mensaje indicativo para el comienzo de la grabación (b) Mensaje indicativo para el termino de grabación

Figura 4.5: Mensajes de interacción para guiar al usuario (3/3).

4.1.2. Aspectos técnicos de la aplicación

En esta sección se abordan aspectos técnicos de la implementación de la aplicación Android. En la Subsección 4.1.2.1 se describe la generación de los archivos de audio que contienen la petición del usuario, seguida de la Subsección 4.1.2.2 en donde se explica la implementación realizada para el envío de las peticiones al micro-servicio. Posteriormente, en la Subsección 4.1.2.3 se detalla la implementación para el despliegue y la lectura mediante voz sintética de los mensajes recibidos por el micro-servicio. Finalmente, en la Subsección 4.1.2.4 se aborda el tema referente a la bitácora que se genera para llevar un control sobre los fallos de la aplicación.

4.1.2.1. Generación de archivos de audio

Para la generación de los archivos de audio, que contienen las peticiones de los usuarios, se utilizó la clase `MediaRecorder` del entorno de desarrollo Android Studio. Esta clase permite a los desarrolladores controlar la captura de medios en un dispositivo, configurar

los parámetros de la grabación, como el formato de salida, la fuente de audio o vídeo, y la calidad, así como iniciar y detener la grabación. Para la utilización de esta clase, es requerido el uso del micrófono del *smartphone*, por lo cual es necesario solicitar permiso al usuario. Esto se realiza mostrando un mensaje al usuario, la primera vez que utiliza la aplicación, en el que se le solicitará permitir el uso del micrófono. Una vez se haya concedido el permiso, el usuario puede utilizar la aplicación; en caso de que el usuario no acepte el uso del micrófono, la aplicación no podrá ser utilizada.

La generación de los archivos de audio sigue el patrón de diseño *Factory*, por lo cual se utiliza una interfaz denominada `AudioRecorder` y tres clases. La primera clase denominada `CreateAudioFileFactory` se encarga de la generación de las instancias de las otras dos clases que generan los archivos de audio en formato MP3 o MP4. Estas clases llevan por nombre `AudioMP3` y `AudioMP4`, respectivamente. Estas clases están conformadas por un constructor en donde se definen los atributos del archivo de audio, como la frecuencia de muestreo (*sampling*, número de canales utilizados (*channel*, nombre del archivo a generar (*filename*, entre otros. Los valores de los parámetros que constituyen estas clases se pueden observar en la Figura 4.6.

```
public AudioMP3(String filename){
    this.sampling = 44100;
    this.bitRate = 64000;
    this.channel = 1;
    this.record = new MediaRecorder();
    this.filename = filename + MainActivity.getAppContext().getString(R.string.audioFile_MP3);
    prepareRecord();
}
```

Figura 4.6: Constructor de la clase `AudioMP3` donde se definen los atributos de la clase

Los métodos empleados en estas clases para la generación de archivos de audio facilitan la gestión integral de estos archivos, mediante métodos que abarcan su preparación, inicialización, cierre y eliminación. El método fundamental para preparar el archivo de audio se denomina `prepareRecord`. En este método, se establecen los parámetros necesarios para crear una instancia de la clase `MediaRecorder` proporcionada por Android Studio.

Entre estos parámetros se incluyen los atributos previamente mencionados, junto con algunos adicionales como: formato de salida, tipo de codificación de audio y fuente de entrada de audio. Estas características se definen mediante los métodos `setOutputFormat`, `setAudioEncoder` y `setAudioSource`, respectivamente.

```
@Override
public void prepareRecord() {
    this.record.setAudioChannels(this.channel);
    this.record.setAudioSource(MediaRecorder.AudioSource.VOICE_RECOGNITION);
    this.record.setOutputFormat(MediaRecorder.OutputFormat.THREE_GPP);
    this.record.setOutputFile(this.filename);
    this.record.setAudioEncoder(MediaRecorder.AudioEncoder.HE_AAC);
    this.record.setAudioSamplingRate(this.sampling);
    this.record.setAudioEncodingBitRate(this.bitRate);

    try{
        this.record.prepare();
    } catch (IOException e) {
        Logger.getInstance().addLog(LOG_TAG);
    }
}
```

Figura 4.7: Características utilizadas en el método `prepareRecord` para la configuración de una instancia de `MediaRecorder`

Como se observa en la Figura 4.7, el formato de salida implementado para la generación de la instancia de `MediaRecorder` es `THREE_GPP`. Este formato es un estándar comúnmente utilizado para la grabación de audio y vídeo en dispositivos móviles. Es compatible con la mayoría de los reproductores multimedia y es ampliamente aceptado en diferentes plataformas. Por otro lado, el tipo de codificación utilizada es `HE_ACC`. Esta codificación es una versión avanzada del estándar AAC (*Advanced Audio Coding*) que proporciona una alta eficiencia de compresión, sin sacrificar significativamente la calidad de audio. Finalmente la fuente de entrada utilizada es `VOICE_RECOGNITION`, la cual está diseñado para la captura de audio destinado al reconocimiento de voz, optimizando la grabación para este propósito, lo cual es beneficioso debido a que el micro-servicio implementado, cuenta con un módulo de identificación de usuarios, mediante el reconocimiento de voz.

La clase `CreateAudioFileFactory` está asociada al botón con icono de micrófono, encargado de la generación de la petición del usuario. Dicha clase también está asociada a las opciones de tipo audio desplegadas cuando se presiona el icono de menú de opciones, que está colocado en la parte superior derecha de la aplicación. Cuando se presiona alguno de los botones referente al tipo de formato a utilizar, se asigna el valor correspondiente a la variable `audioFormat`, la cual es encargada de definir el tipo de archivo de audio a generar, mediante la clase `CreateAudioFileFactory`. Posteriormente, cuando se presiona el botón de micrófono, se genera la instancia del archivo de audio. La Figura 4.8 muestra cómo se implementa esta asociación entre el botón y la clase generadora de los archivos de audio.

```
if(!newUser){
    if(Objects.equals(this.audioFormat, b: "mp3")) {
        request = CreateRequestFactory.createRequest( format: "user", filename: this.filename+".mp3",text);
    } else request = CreateRequestFactory.createRequest( format: "user", filename: this.filename+".mp4",text);
    request.makeRequest();
}
```

Figura 4.8: Implementación de la clase `CreateAudioFileFactory` para la generación de archivos de audio

4.1.2.2. Envío de peticiones

Para la generación de las peticiones del usuario, se han creado tres clases y una interfaz basadas en el patrón de diseño *Factory*. Estas clases son `CreateRequestFactory`, `NewUserRequest` y `UserRequest`. Estas dos ultimas utilizan la interfaz `Request` para la implementación de dos métodos denominados `prepareRequest` y `makeRequest`. Las diferencias entre las clases `NewUserRequest` y `UserRequest` residen en el envío de las peticiones y los *endpoints* utilizados (*newurser* y *whisper*, que se encuentran en los archivos de configuración dependiendo del idioma). Respecto a la clase `NewUserRequest`, se envían todas las muestras de audio solicitadas para el registro del usuario, haciendo sólo una petición al micro-servicio, mediante el método POST. Por otra parte, respecto a la petición generada por la clase `UserRequest`, sólo se envía un archivo de audio que contiene la petición del usuario. Para el envío de las peticiones utilizando el protocolo de comunicación HTTP, se utilizó la biblioteca `OkHttpClient`, que permite el envío de peticiones por cualquier método utilizado por el protocolo de comunicación, generando un hilo secundario en la aplicación. De esta manera, se puede seguir utilizando alguna otra funcionalidad de la aplicación, mientras se lleva a cabo el envío de la petición del usuario al micro-servicio, sin la necesidad de tener que esperar a la respuesta del micro-servicio.

Estas dos clases cuentan con los métodos `onFailure` y `onResponse`, los cuales tienen como funcionalidad, realizar alguna acción dependiendo del evento sucedido. El método `onFailure` se encarga de enviar un mensaje al usuario, indicándole si existió algún inconveniente con la comunicación entre la aplicación Android y el micro-servicio. En el caso de no poder realizar la comunicación con el micro-servicio, la aplicación envía al usuario un mensaje de "Error en la conexión con el micro-servicio. Intente más tarde", como se muestra en la Figura 4.9.

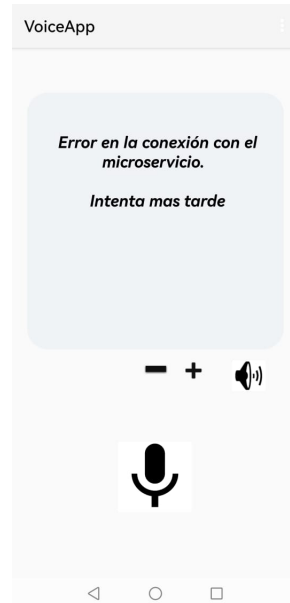


Figura 4.9: Mensaje de error de conexión con el micro-servicio desplegado en el cuadro de texto de la aplicación.

Por otro lado, el método `onResponse` se encarga de obtener las respuestas generadas por el micro-servicio para mostrarlas al usuario, ya sea si se trata de un mensaje de notificación para confirmar el procesamiento de su petición o para indicarle que su petición ha sido recibida, pero no ha podido ser procesada por el micro-servicio. Este método se puede visualizar en la Figura 4.10. Además, este método también contiene una instancia de la clase `Text`, encargada del despliegue de todos los mensajes de texto en la aplicación, utilizando el cuadro de texto localizado al centro de la aplicación. La explicación de esta clase se encuentra en la Sección 4.1.2.3.

```
@Override
public void onResponse(@NonNull Call call, @NonNull Response response) throws IOException {
    if (response.isSuccessful()){
        String responseData = response.body().string();
        Text message = new Text(text);
        message.showMessage(responseData);
    }
}
```

Figura 4.10: Implementación del método `onResponse` para recibir las respuestas del micro-servicio

4.1.2.3. Despliegue y lectura de mensajes de interacción con el usuario

Como se mencionó anteriormente, la aplicación cuenta con dos funcionalidades que permiten al usuario interactuar con el micro-servicio. Estas funcionalidades son el despliegue de mensajes de texto y mediante la lectura de estos mensajes utilizando voz sintética. Para el despliegue de estos mensajes de texto en la aplicación, se utiliza la clase `Text`.

La clase `Text` contiene los métodos necesarios para la manipulación de estos mensajes de texto. Dentro de los métodos con los que cuenta son: `showMessage`, `readMessage`, `makeBigger` y `makeSmaller`. Estos métodos a su vez hacen uso de la clase `TextView` contenida dentro del entorno de Android Studio. Esta clase permite la manipulación del texto para su despliegue y, en este caso, el cambio en el tamaño del texto. Para el despliegue del mensaje en el cuadro de texto, se utiliza la función `setText` en el método `showMessage` de la clase `Text`. Así mismo, para la manipulación del tamaño de la fuente del texto, se utilizan las funciones `getTextSize` y `scaleDensity`. Ambas funciones generan una variable entera correspondiente al tamaño de la fuente del texto. Una vez que se obtiene esta variable, esta se incrementa o decrementa mediante los métodos `makeBigger` y `makeSmaller`, respectivamente. La implementación de estos métodos se puede visualizar en la Figura 4.11.

```
protected void makeBigger(){
    float currentSize = text.getTextSize()/ MainActivity.getAppContext().getResources().getDisplayMetrics().scaledDensity;
    float nextSize = currentSize + 1;
    text.setTextSize(nextSize);
}

1 usage
protected void makeSmaller(){
    float currentSize = text.getTextSize()/ MainActivity.getAppContext().getResources().getDisplayMetrics().scaledDensity;
    float nextSize = currentSize - 1;
    text.setTextSize(nextSize);}
}
```

Figura 4.11: Implementación de los métodos `makeBigger` y `makeSmaller` para la manipulación de la fuente de los mensajes de texto

Para la lectura de los mensajes desplegados en el cuadro de texto de la aplicación, se utiliza la clase `Reader`, la cual utiliza la clase `TextToSpeech` que contiene Android Studio. Esta clase permite la lectura de textos escritos en distintos idiomas, dependiendo de la configuración establecida. Para este caso, se definen dos configuraciones para la lectura de los textos en idioma español e inglés. La explicación de estas configuraciones se mencionan en párrafos posteriores.

La clase `Reader` cuenta con tres métodos. El primero de ellos es el método de inicialización `onInit` en donde se define el lenguaje a utilizar mediante la función `Locale`.

Esta función recibe, como parámetros de entrada, el lenguaje a utilizar y el país, ya que existen variaciones en el mismo idioma (e.g., el inglés de Estados Unidos difiere en algunas palabras con el inglés de Reino Unido). Para la definición de los idiomas a utilizar, los parámetros de entrada son: `es,MEX`, para el idioma español y `en,GB`, para el idioma inglés. Los parámetros mencionados se encuentran en el archivo de configuración `strings`.

El método `onInit` inicializa una instancia de la clase `TextToSpeech` de Android Studio y verifica si se ha inicializado correctamente mediante la función `TextToSpeech.SUCCESS`. Si la instancia de esta clase se ha inicializado correctamente se procede a la definición del idioma a utilizar. Por otro lado, si por algún motivo la instancia de la clase `TextToSpeech` no se ha podido inicializar, se genera un registro en la bitácora de fallos mediante el método `getInstance` de la clase `Logger`, la cual se mencionará en párrafos posteriores. Estos registros se realizan con el fin de ayudar al desarrollador a identificar, de manera más rápida, los fallos de la aplicación para poder corregirlos. La implementación de este método se puede visualizar en la Figura 4.12.

```

1 related problem
private void init(){
    try{

        tts = new TextToSpeech(this.context, new TextToSpeech.OnInitListener() {
            @Override
            public void onInit(int status) {

                if (status == TextToSpeech.SUCCESS) {
                    Locale language = new Locale(language, country);
                    int result = tts.setLanguage(language);
                    isloaded = true;

                    if (result == TextToSpeech.LANG_MISSING_DATA || result == TextToSpeech.LANG_NOT_SUPPORTED) {
                        Logger.getInstance().addLog(LANGUAGE_FAIL);
                    }
                } else {
                    Logger.getInstance().addLog(TTS);}
            }
        });
    }catch(Exception e){
        e.printStackTrace();
    }
}

```

Figura 4.12: Implementación del método `onInit` para la inicialización de una instancia de la clase `Reader`.

Una vez que ya se ha inicializado correctamente, se utiliza el método `initQueue`, el cual carga y lee el texto mediante la función `TextToSpeech.speak`, a la cual se le ingresa como parámetro el texto a leer. Si existe algún inconveniente durante este proceso, se genera de

igual manera un registro en la bitácora de fallos, mediante el método `getInstance` de la clase `Logger`.

4.1.2.4. Bitácora de fallos

La aplicación Android cuenta con una bitácora para llevar un registro sobre los fallos que lleguen a suceder durante su ejecución. Esta bitácora se genera mediante la clase `Logger`, siguiendo el patrón de diseño *Singleton* para generar sólo una instancia de esta clase y todos los reportes generados permanezcan en la misma instancia. Esta clase cuenta con un constructor, así como los métodos `getInstance`, `addLog` y `closeLogger`. El constructor de esta clase se encarga de crear la instancia requerida para la generación de la bitácora, mientras que el método `getInstance` se encarga de no generar más de una instancia de la clase `Log`, ya que sólo en caso de que no exista una instancia de esta clase, se genera; de otro modo, dicho método utiliza la instancia que ya ha sido creada. Esto se puede visualizar en la Figura 4.13.

```
public static Logger getInstance(){
    if(instance == null){
        instance = new Logger();
    }

    return instance;
}
```

Figura 4.13: Implementación del método `getInstance` para la generación del patrón de diseño *Singleton* de la clase `Logger`.

4.2. Sistema de captura permanente

A continuación, se presenta la implementación realizada para el sistema de captura permanente. En la Sección 4.2.1 se presentan la elección del *hardware* utilizado para la implementación del sistema, seguida de la Sección 4.2.2 donde se mencionan las herramientas utilizadas. Posteriormente, se aborda en la Sección 4.2.3 los aspectos técnicos del desarrollo del sistema y en la Sección 4.2.4 se detalla el estudio realizado para la selección de la palabra de activación. Finalmente, en la Sección 4.2.5 se presenta el análisis del rango de escucha del sistema de captura permanente.

4.2.1. Hardware seleccionado

A continuación, se presentan los análisis realizados para la selección del *hardware* utilizado para implementar el sistema de captura permanente. En la Sección 4.2.1.1 se trata el tema de la selección del dispositivo de bajo consumo a utilizar, seguido de la Sección 4.2.1.2 en donde se define el micrófono a utilizar para la captura de audio.

4.2.1.1. Dispositivo de bajo consumo

Para la selección de los dispositivos de bajo consumo a utilizar, como medios captadores de audio, se tomaron en consideración cuatro, los cuales son: ESP32, ESP32-S3, Raspberri Pico y Arduino UNO. Las características a consideración se basaron en las necesidades para la captura y envío de la información al micro-servicio. Entre estas características se encuentran: el tipo de procesador, frecuencia de trabajo, conectividad inalámbrica, entre otras, las cuales se pueden observar en la Tabla 4.1.

Tabla 4.1: Comparativa de dispositivos de bajo consumo para selección de dispositivos de entrada del sistema

Características	ESP32S2	ESP32S3	RBPI Pico	Arduino UNO
Procesador	32-bit Xtensa LX7 un solo núcleo	32-bit Xtensa LX7 de doble núcleo	RP2040 de doble núcleo	ATMEGA328P
Frecuencia	240 MHz o mayor	240 MHz o mayor	133 MHz o mayor	4/10/20 MHz
SRAM	320KB	512KB	264KB	2KB
ROM	128KB	384KB	-	32KB
Wi-fi	Wi-Fi 4	Wi-Fi 4	No incluido	No incluido
Bluetooth	Bluetooth 5.0	Bluetooth 5.0	No incluido	No incluido
Plataforma	Arduino	Arduino	Raspbian	Arduino
Lenguaje de programación	JavaScript y C++	JavaScript y C++	C++ y MicroPython	C++

La comparativa detallada en la Tabla 4.1 resalta las características sobresalientes de los dispositivos ESP32 y ESP32-S3 en comparación con Raspberry Pico y Arduino UNO. Los últimos dos dispositivos presentan limitaciones significativas en términos de memoria RAM, un componente esencial para el desarrollo del sistema junto con la conectividad

inalámbrica, ya que se pretende la utilización del protocolo de comunicación HTTP. En el caso de Raspberry Pico y Arduino UNO, estos carecen de módulos de Wi-Fi, lo que los descarta para la implementación del sistema de captura permanente que requiere esta funcionalidad.

En la comparativa específica entre ESP32 y ESP32-S3, se destaca que ESP32-S3 ofrece mejoras significativas, como una mayor capacidad de memoria RAM y un procesador de doble núcleo. Estas características son esenciales para el desarrollo del sistema, donde la programación multitarea es esencial. La presencia de más núcleos en ESP32-S3 mejora significativamente el rendimiento y la eficiencia en la ejecución de tareas.

Con base en estas consideraciones, la elección de utilizar el dispositivo ESP32-S3 se sustenta en su capacidad para cumplir de manera óptima las necesidades del sistema, ofreciendo un equilibrio adecuado entre recursos, conectividad y potencia de procesamiento.

4.2.1.2. Micrófono para captura de audio

Para la selección del dispositivo encargado de la captura de audio, se tomaron en consideración dos micrófonos, el primero es el micrófono analógico Adafruit AGC-MAX9814 [integrated, 2014] y el micrófono digital INMP441 [InvenSense, 2014]. Para la determinación del micrófono a utilizar, se realizó una comparativa mostrada en la Tabla 4.2, sobre sus características, basadas en sus hojas de especificaciones.

Tabla 4.2: Comparativa entre los micrófonos Adafruit AGC-MAX9814 y INMP441

Característica	Adafruit AGC-MAX9814	INMP441
Interfaz de comunicación	Analógica	I2S o PDM
Rango de voltaje de operación	3.3V - 5V	1.8V - 3.3V
Sensibilidad ajustable	Sí	No
Ganancia ajustable	Sí	No
Respuesta de frecuencia	20 Hz - 20 kHz	50 Hz - 15 kHz
Consumo de energía	Bajo	Bajo

En la fase inicial, se optó por utilizar el micrófono Adafruit AGC-MAX9814 debido a su amplio rango de frecuencia y su flexibilidad para ajustar la sensibilidad, mediante la configuración de la ganancia en decibelios (dB) entre 60dB, 50dB y 40dB. Esta versatilidad permitía adaptar el rango de escucha del micrófono a las necesidades del sistema. Se llevaron a cabo pruebas de las diferentes configuraciones de sensibilidad, mostrando la amplitud de la señal en un osciloscopio. Los resultados se pueden observar en las Figuras 4.14a, 4.14b y 4.15.



(a) Señal de audio a 40dB capturada por el micrófono Adafruit AGC-MAX9814 (b) Señal de audio a 50dB capturada por el micrófono Adafruit AGC-MAX9814

Figura 4.14: Señales de audio a 40dB y 50dB utilizando el micrófono Adafruit AGC-MAX9814



Figura 4.15: Señal de audio a 60dB capturada por el micrófono Adafruit AGC-MAX9814

El aumento de la amplitud de la señal de audio capturada al utilizar una configuración de 60dB indica la capacidad del micrófono para recoger información más detallada del entorno, lo que facilita la captura de sonidos a distancias mayores. Esto resultaba beneficioso para ampliar el rango de escucha del sistema.

Sin embargo, al almacenar la información capturada por el micrófono en archivos de audio, los resultados no cumplieron con las expectativas. Al reproducir los archivos en cualquier configuración de decibelios, se evidenció un exceso de ruido proveniente del entorno y de interferencias generadas por los propios dispositivos utilizados (ESP32-S3, cables de conexión y el micrófono). Además, mientras más se aumentaban los decibelios, mayor era el ruido capturado. Esto se debe a que, al ser el sistema más sensible, permitiendo mayor rango en la distancia de captura de audio, también permitía capturar mayor ruido. Esto resultó en una reproducción de audio dominada por el ruido, sin una clara distinción de la información deseada.

Ante la primera limitación identificada, se implementaron medidas para eliminar el ruido en los archivos de audio capturados. Se utilizaron capacitores de 10 microfaradios y resistencias variables, con el fin de generar un filtro pasa bajas analógico para eliminar las interferencias y mejorar la calidad de la señal de audio. La implementación de estos elementos puede observarse en la Figura 4.16.

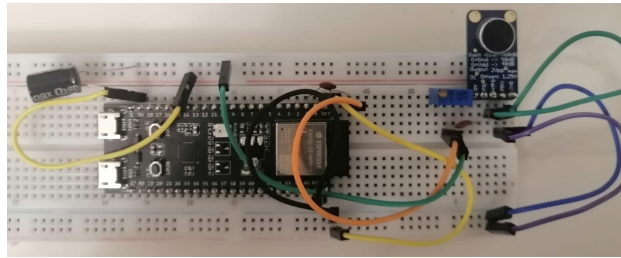


Figura 4.16: Diseño de filtro analógico pasa bajas para eliminación de ruido

Si bien se logró disminuir casi en su totalidad el ruido de los archivos de audio, surgió una segunda limitación al constatar que la implementación del micrófono no permitía la captura de información, sin involucrar la unidad de procesamiento principal (CPU). Esta limitación imposibilitaba la ejecución efectiva de multitareas, comprometiendo la capacidad del sistema para llevar a cabo procesos concurrentes de manera eficiente. Por esta razón, se tomó la decisión de descartar el uso de este micrófono.

En contraste, el micrófono INMP441, al ser de naturaleza digital, demostró ser más eficiente en el manejo de la información capturada. La implementación del protocolo I2S (Inter-IC Sound) facilitó la transmisión de datos de audio digital, eliminando las interferencias analógicas y garantizando una transmisión de señal más clara y libre de pérdidas. Además, este micrófono tiene la capacidad de utilizar, mediante el protocolo I2S, el modo de Acceso Directo a Memoria (DMA, por sus siglas en inglés), el cual permite una transferencia de datos sin intervención constante de la CPU, permitiendo llevar a cabo múltiples tareas, mientras se mantiene la captura del audio mediante el micrófono.

En resumen, la elección del micrófono INMP441 se respalda en su capacidad digital y en la utilización del protocolo I2S Y DMA. Estas características no sólo garantizan una calidad de audio superior, sino la necesidad de utilizar algún filtro digital o analógico, sino que también optimiza el uso de los recursos del ESP32-S3, contribuyendo al rendimiento general del sistema de captura permanente de manera significativa. La Figura 4.17 muestra la implementación del *hardware* del sistema de captura permanente.

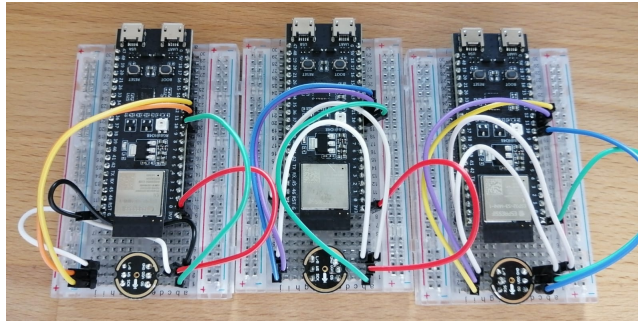


Figura 4.17: *Hardware* utilizado en la implementación del sistema de captura permanente

4.2.2. Frameworks y bibliotecas utilizadas

Se decidió utilizar el marco de desarrollo `Arduino IDE`, basado en el lenguaje de programación `C++`. Esta elección permite implementaciones más simples y prácticas, es compatible con el dispositivo utilizado y cuenta con las bibliotecas necesarias para implementar la solución propuesta y aprovechar los recursos del dispositivo.

`ESP32-S3` es un dispositivo multi-núcleo que tiene dos unidades de procesamiento, permitiendo realizar operaciones simultáneas en tiempo real. Para aprovechar esta capacidad, se utilizó `FreeRTOS`, un sistema operativo de código abierto adecuado para dispositivos `ESP-ID` como el `ESP32-S3`. Facilita implementaciones multi-núcleo, concurrentes o combinadas, dependiendo de las capacidades del dispositivo [Fre,]. Dentro del marco de desarrollo `Arduino IDE`, es posible utilizar este núcleo, creando tareas que pueden ejecutarse de manera multi-núcleo y concurrente.

Como se mencionó anteriormente, el micrófono utilizado opera bajo el protocolo de comunicación *Inter-Integrated Circuit Sound* (`I2S`), que permite la transmisión y recepción de datos de audio digital entre dispositivos. Este protocolo de comunicación brinda acceso directo a la memoria del dispositivo sin involucrar a la CPU, mejorando la eficiencia al realizar tareas concurrentes o multi-núcleo .

El marco de desarrollo `Arduino IDE` facilita la utilización de este protocolo, a través de la biblioteca `<driver/i2s.h>`. Para emplear este protocolo, se requiere una configuración inicial donde se definen los pines para la comunicación `I2S`. Posteriormente, se especifica la configuración de la interfaz, incluyendo parámetros como la frecuencia de muestreo, bits por muestra de audio, formato de datos, longitud del *buffer*, número de *buffer*, entre otros. Una vez completada la configuración, se realiza la inicialización para comenzar el proceso de captura de audio.

Finalmente, para la transmisión de los *buffers* de audio, a través del protocolo de

comunicación HTTP, se emplearon las bibliotecas `<WiFi.h>` y `<HTTPClient.h>`. Estas bibliotecas permiten la conectividad a la red Wi-Fi y la transmisión de datos, respectivamente. La información necesaria para utilizar estas bibliotecas incluye el nombre de la red a la cual se conectará el dispositivo, la contraseña de la red y las direcciones IP de destino a las cuales se enviarán los datos.

4.2.3. Aspectos técnicos del sistema

En la solución generada para la implementación del sistema de captura permanente, se tuvo que tener en consideración la capacidad de memoria del dispositivo, que está limitada a 512 KB [Espressif, 2023]. Esto implica que la asignación dinámica de memoria fue necesaria para utilizar eficientemente la memoria del dispositivo en su máxima capacidad. Como resultado, el sistema de captura permanente se compone de tres tareas principales y una tarea secundaria: `WakeWordTask`, `AudioRecorderTask`, `PostTask` y `LigtherTask`, respectivamente.

Estas tareas deben ejecutarse, de manera concurrente, en los núcleos de ESP32-S3, ya que el sistema debe monitorear continuamente si el usuario ha pronunciado la palabra de activación mientras captura sonido ambiente simultáneamente. Una vez identificado, inicia la grabación de la solicitud mientras activa el LED para indicar al usuario que la solicitud se está capturando y posteriormente la envía al micro-servicio.

Con el fin de evitar *hard coding*, se generó un archivo de configuración del sistema denominado `Config.h`. Este archivo contiene todos los parámetros necesarios para el correcto funcionamiento del sistema. Entre los parámetros que lo conforman se encuentran: nombre y contraseña de la red Wi-Fi a utilizar, *endpoints* utilizados para el envío de la información al micro-servicio y pines del dispositivo de bajo consumo necesarios para la conexión del micrófono digital. La Figura 4.18 muestra parte del archivo de configuración mencionado.

```
ino  Config.h
#define WIFI_SSID "labnet"
#define WIFI_PSWD "inthrustwetrust"

//Labnet-IP
#define WAKEWORD_URL "http://10.2.2.153:5000/wakeword"
#define COMAND_URL "http://10.2.2.153:5000/esp32"

//Microphone settings
#define I2S_WS GPIO_NUM_11
#define I2S_SD GPIO_NUM_10
#define I2S_SCK GPIO_NUM_12
#define I2S_PORT I2S_NUM_0
```

Figura 4.18: Archivo de configuración del sistema de captura permanente

4.2.3.1. Inicialización del sistema

Debido a que se ha utilizado el IDE de Arduino como plataforma de desarrollo, la inicialización del sistema se lleva a cabo mediante la función `setup`.

Previo a la inicialización del sistema es importante definir la configuración para el uso del micrófono digital. Para su configuración, se requiere la especificación de los pines de conexión con el dispositivo ESP32-S3, los cuales se encuentran definidos en el archivo de configuración `Config.h`. Los pines fueron configurados con base en la documentación del dispositivo ESP32-S3 [Espressif, 2023], definiendo así los pines 11, 10, 12 y 0 para la comunicación con el micrófono digital.

Una vez definidos los pines de conexión, se requiere la configuración del protocolo de comunicación I2S, en donde se definen parámetros como: modo de uso (`i2s_mode_t`), tasa de muestreo a utilizar para la captura del audio (`sample_rate`), cantidad de bits utilizados por muestra de audio (`bits_per_sample`), canal del micrófono a utilizar (`channel_format`), formato de comunicación (`communication_format`), cantidad de *buffer* DMA a utilizar (`dma_buf_count`) y longitud del *buffer* DMA (`dma_buf_lendma_buf_len`).

Los parámetros esenciales a tomar en consideración fueron la frecuencia de muestreo y la cantidad de bits por muestra de audio, ya que al tener una memoria limitada, estos parámetros influyen de manera significativa en el rendimiento del sistema, limitando el tiempo de grabación que es posible obtener dependiendo de la configuración establecida.

Para definir los parámetros óptimos, se optó inicialmente por una frecuencia de muestreo de 44.1 KHz y 16 bits por muestra, la cual es reconocida por su uso habitual en la obtención de calidad de audio estándar. Esta frecuencia de muestreo, además, es un estándar comúnmente empleado en grabaciones de audio para CD. A partir de esta frecuencia, se calculó la cantidad de memoria requerida por segundo de grabación, utilizando la ecuación 4.1:

$$\text{Memoria por segundo} = 44,1 \text{ KHz} \times 1 \times \left(\frac{16}{8}\right) = 88,2 \text{ KB} \quad (4.1)$$

El resultado indicó que son necesarios 88.2 KB de memoria por cada segundo de audio grabado. Luego, se calculó mediante la ecuación 4.2, el tiempo máximo de grabación posible con el dispositivo ESP32-S3, considerando su capacidad de memoria disponible:

$$\text{Tiempo máximo de grabación} = \frac{514 \text{ KB}}{88,2 \text{ KB}} = 5,82 \text{ seg} \quad (4.2)$$

Los resultados teóricos indicaron que, con una frecuencia de muestreo de 44.1 KHz, el dispositivo podría generar grabaciones de apenas cinco segundos, lo cual es una duración insuficiente para los objetivos de este sistema. Además, estos resultados son ideales y asumen que toda la memoria está disponible, lo cual no es realista, ya que el código en

sí mismo ocupa memoria al declarar variables y realizar procesos, dejando en claro que el tiempo de grabación real disponible se encontraba por debajo de los cinco segundos. Por lo tanto, esta configuración no es una opción a considerar.

Con base en los cálculos realizados, la frecuencia de muestreo y la cantidad de bits por muestra seleccionados fueron de 16 KHz y 16 bits, respectivamente, ya que al utilizar esta configuración, los resultados obtenidos de las ecuaciones 4.3 y 4.4, muestran que idealmente es posible realizar grabaciones de aproximadamente 16 segundos, lo cual nos indica que, en términos reales, después de realizar algunas pruebas, el tiempo aproximado de grabación es de 9 segundos:

$$\text{Memoria por segundo} = 16 \text{ KHz} \times 1 \times \left(\frac{16}{8}\right) = 32 \text{ KB} \quad (4.3)$$

$$\text{Tiempo máximo de grabación} = \frac{514 \text{ KB}}{32 \text{ KB}} = 16,06 \text{ seg} \quad (4.4)$$

Por otra parte si se optaba por reducir aún más la frecuencia de muestreo o la cantidad de bits por muestra, la calidad del audio se vería mayormente afectada, influyendo en procesos futuros como la traducción del archivo de audio a texto por el modulo de procesamiento de lenguaje natural del micro-servicio, ya que al ser menor la calidad del audio, la precisión en la traducción del archivo de audio también se ve disminuida. La configuración realizada para la inicialización del micrófono digital, se puede visualizar en la Figura 4.19.

```
void i2s_install() {
    // Set up I2S Processor configuration
    const i2s_config_t i2s_config = {
        .mode = i2s_mode_t(I2S_MODE_MASTER | I2S_MODE_RX),
        .sample_rate = AUDIO_SAMPLE,
        .bits_per_sample = i2s_bits_per_sample_t(16),
        .channel_format = I2S_CHANNEL_FMT_ONLY_LEFT,
        .communication_format = i2s_comm_format_t(I2S_COMM_FORMAT_STAND_I2S),
        .intr_alloc_flags = 0,
        .dma_buf_count = 64, //Number of buffer created
        .dma_buf_len = BUFFER_LEN,
        .use_apll = false};

    i2s_driver_install(I2S_PORT, &i2s_config, 0, NULL);
}
```

Figura 4.19: Parámetros de configuración del protocolo I2S para la inicialización del micrófono digital.

Habiendo configurado los requerimientos necesarios para la inicialización del micrófono digital, la función `setup` inicializa el sistema realizando primeramente la conexión Wi-Fi, seguida de la inicialización del micrófono digital para finalmente inicializar la `cola de peticiones` y las tareas generadas para el correcto funcionamiento del sistema, de las cuales se hablará en párrafos posteriores.

4.2.3.2. Implementación de tareas

Como se mencionó al principio de esta sección, el sistema está basado en la ejecución de tareas debido a la necesidad de realizar acciones simultáneas como la grabación de audio y el envío de peticiones. Por esta razón, las tareas fueron creadas y ejecutadas, de manera concurrente, utilizando `FreeRTOS`, el cual es un kernel de sistema operativo en tiempo real que facilita la ejecución eficiente de tareas en sistemas embebidos y aplicaciones de tiempo real.

Cada tarea fue generada mediante la función `xTaskCreatePinnedToCore` que permite la generación y ejecución de dichas tareas, con base en un grado de importancia que el desarrollador asigna. Mientras más grande sea el valor, mayor importancia tendrá la tarea y, por consiguiente, tendrá mayor prioridad para ser ejecutada. Así mismo, esta función requiere la definición de otros parámetros como el núcleo en que se llevará a cabo la tarea, en el caso de que el dispositivo donde se implemente el código cuente con más de uno. Además, es necesario la asignación de memoria a la tarea que se llevará a cabo, un identificador y un orquestador de tareas, el cual servirá para detener o iniciar tareas dentro de otras.

Para la implementación de las tres tareas principales, (*WakeWordTask*, *AudioRecorderTask* y *PostTask*), se asignó una memoria de 2.5 KB a cada una debido a las acciones específicas que llevan a cabo, como la generación de *buffers* de audio y el envío de peticiones. Por otro lado, se destinaron 1.5 KB de memoria a la tarea encargada del encendido y apagado del LED indicador. Con el objetivo de facilitar un manejo eficiente del envío de los *buffers* de audio al micro-servicio, se tomó la decisión de asignar un núcleo del dispositivo ESP32-S3 exclusivamente para la ejecución de la tarea *PostTask*, mientras que el segundo núcleo se dedicó a la ejecución de las tres tareas restantes.

A pesar de tener tres tareas asignadas, el segundo núcleo se encarga de ejecutar sólo dos de manera concurrente; así, mientras la tarea *WakeWordTask* está activa, la tarea *AudioRecorderTask* se mantiene desactivada y viceversa. La tarea *WakeWordTask* se encarga de generar *buffers* de un segundo de audio para la identificación de la palabra de activación, mientras que la tarea *AudioRecorderTask* se dedica a la captura de la petición del usuario, por lo cual es posible su ejecución de manera simultánea.

En cuanto al grado de importancia, se asignó el mayor grado (grado 3) a la tarea *PostTask*, seguido de las dos tareas *WakeWordTask* y *AudioRecorderTask*. La tarea *LigtherTask* fue definida con el menor grado de importancia, ya que su única función es indicar al usuario de alguna acción a realizar. La estructura y asignación de las tareas se ilustra en la Figura 4.20.

```
xTaskCreatePinnedToCore(WakeWordTask, "WakeWord", 2500, NULL, 2, &WakeWordHandle, 0);
xTaskCreatePinnedToCore(AudioRecorderTask, "RecordAudio", 2500, NULL, 2, &AudioRecordHandle, 0);
xTaskCreatePinnedToCore(PostTask, "SendMessage", 2500, NULL, 3, &PostHandle, 1);
xTaskCreatePinnedToCore(LigtherTask, "Lighter", 1500, NULL, 1, &LighterHandle, 0);
```

Figura 4.20: Generación de las tareas del sistema de captura permanente

La asignación de un sólo núcleo a la tarea encargada del envío constante de peticiones al micro-servicio (*PostTask*) se debe a su importancia crítica para el funcionamiento integral del sistema. Esta tarea desempeña un papel fundamental para el envío de peticiones continuas para la identificación de la palabra de activación y posteriormente para la atención de las solicitudes realizadas por el usuario. En caso de que se produzca algún retraso o inconveniente en el envío de peticiones para la identificación de la palabra de activación, existe el riesgo de acumulación de *buffers* generados por la tarea *WakeWordTask* en la cola de peticiones, del cual se hablará en párrafos siguientes. Este escenario podría provocar un desbordamiento de memoria, generando así un problema crítico en el sistema.

La implementación de la tarea *WakeWordTask*, encargada de la generación de *buffers* de un segundo para la identificación de la palabra de activación, inicia con la lectura del audio capturado por el micrófono digital.

La información de audio capturada por el micrófono se almacena en *buffers* generados mediante el protocolo I2S. Este método evita la necesidad de llamar al CPU cada vez que el micrófono capta un dato de audio. En su lugar, se aprovecha el DMA mencionado en los últimos párrafos de la Sección 4.2.1.2. Este mecanismo guarda la información de audio capturada por el micrófono en *buffers* definidos durante la inicialización del micrófono, mediante las variables *dma_buf_count* y *dma_buf_len*. Cuando un *buffer* se llena, se solicita la intervención del CPU para procesarlo. De este modo, cada vez que la tarea *WakeWordTask* realiza una lectura de audio, accede a toda la información contenida en uno de los *buffers* mencionados anteriormente, mientras el siguiente *buffer* del DMA comienza a llenarse.

Una vez que la tarea *WakeWordTask* lee un *buffer* de audio del DMA, mediante la función *i2s_read*, obtiene el promedio de la información y lo compara con el umbral establecido para identificar cuándo se debe comenzar la generación de los *buffers* de audio de un segundo que se envían al micro-servicio. Este umbral está relacionado con la amplitud de

la señal de audio. Mientras esta amplitud no sea superada, el sistema interpreta que no hay algún usuario hablando y no se continua leyendo la información de audio capturada por el micrófono. Por otro lado, si la amplitud de la señal supera el umbral, el sistema infiere que existe la posibilidad de que alguien requiera solicitar una petición, por lo cual, se comienza la generación de los *buffers* de un segundo, mediante el método `getBuffers`. Este método es el encargado de la generación de *buffers* mediante el uso de memoria dinámica ya que, debido a las limitaciones de memoria, es fundamental el manejo de esta. La implementación de la tarea mencionada puede visualizarse en la Figura 4.21.

```

void WakeWordTask(void* param){
    size_t bytesIn = 0;
    float mean =0;

    while(1){

        esp_err_t result = i2s_read(I2S_PORT, &sBuffer, BUFFER_LEN, &bytesIn, portMAX_DELAY);
        if (result == ESP_OK){
            uint32_t samples_read = bytesIn / 8;
            if (samples_read > 0) {
                mean = getMean(bytesIn, mean, samples_read);
                if (mean > THRESHOLD){
                    getBuffers(buffers, WAKEWORD_TIME, samples_read);
                }
            }
            vTaskDelay(pdMS_TO_TICKS(10));
        }
    }
}

```

Figura 4.21: Implementación de la tarea `WakeWordTask`

La tarea `AudioRecorderTask` utiliza los mismos métodos para la generación del *buffer* que contiene la petición del usuario, variando solamente en tiempo, ya que para la petición del usuario se genera un *buffer* correspondiente a seis segundos de audio. Sin embargo, esta tarea varía en la forma de implementarse.

Como se mencionó, las tareas `WakeWordTask` y `AudioRecorderTask` no deben ejecutarse simultáneamente. Por esta razón, al momento de inicializar la tarea `AudioRecorderTask` en la función `setup`, esta se suspende a si misma, mediante la función `vTaskSuspended`, garantizando que sólo la tarea `WakeWordTask` se mantenga en ejecución. Una vez que esta es activada por la tarea `PostTask`, se genera un *buffer* de seis segundos mediante el uso de memoria dinámica y, al mismo tiempo, la tarea `AudioRecorderTask` activa la tarea `LigtherTask` encargada de encender el LED indicador, mediante la función `vTaskResume`, para informar al usuario que puede comenzar a realizar su petición, como se muestra en la Figura 4.22.

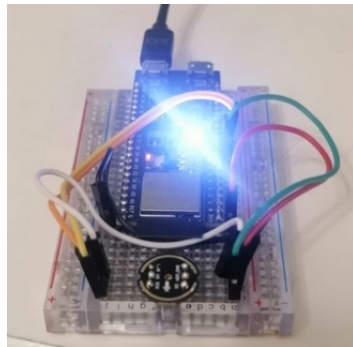


Figura 4.22: Encendido del LED indicador para comenzar a grabar la petición

Al inicializarse la tarea `LigtherTask`, se comienza con la captura de la petición del usuario, mediante el método `fillBuffer`, para posteriormente ser enviado a la cola de peticiones mediante la función `xQueueSend`. Para fines demostrativos, cada que se comienza a grabar la petición del usuario, se muestra en pantalla un mensaje de *Recording user request...*

La tarea `LigtherTask` sólo se encarga de encender y apagar el LED indicador. Sin embargo, para evitar que se ejecute desde el inicio, al igual que la tarea `AudioRecorderTask`, contiene la función `vTaskSuspend` que la mantiene en modo de espera hasta ser inicializada de nuevo, mediante la función `vTaskResume` usada en la tarea `AudioRecorderTask`. La tarea `LigtherTask` es utilizada en la tarea `AudioRecorderTask` y contiene los métodos `ledOn` y `ledOff`. El método `ledOn` recibe el tiempo en microsegundos en que el LED debe permanecer encendido y el método `ledOff` recibe como parámetro el tiempo en microsegundos en que el LED debe permanecer apagado, como se muestra en la Figura 4.23.

```
void LigtherTask(void* param){
    vTaskSuspend(LigtherHandle);
    while(1){
        ledOn(time);
        ledoff(time);
    }
}
```

Figura 4.23: Tarea `LigtherTask` encargada del manejo del LED indicador

Finalmente, la tarea `PostTask`, encargada del envío de las peticiones al micro-servicio, tanto para la identificación de la palabra de activación como para las solicitudes de los usuarios, también es la encargada de la orquestación de la suspensión de la tarea `WakeWordTask` y de la activación de la tarea `vTaskResume`.

La tarea `PostTask` contiene la función `QueueReceive` con la cual detecta si existe algún elemento dentro de la Cola de peticiones. Así mismo, contiene una variable de tipo `bool` que sirve como bandera, denominada `flag_task`, que indica dependiendo de su estado, la tarea que se encuentra activada, siendo `true` para la tarea `WakeWordTask` y `false` para la tarea `AudioRecorderTask`.

La tarea `PostTask` se mantiene siempre activa, a la espera de elementos que se encuentren en la Cola de peticiones. Una vez detectado algún elemento en dicha cola, se verifica la bandera. En caso de tener un valor positivo, se envía el `buffer` al micro-servicio, mediante el método `setBuffer`, ingresando como parámetros de entrada, el `buffer` detectado en la Cola de peticiones, seguido del tiempo de audio contenido en el `buffer` y el `endpoint` indicado para la comunicación con el micro-servicio. Posteriormente, se espera la respuesta del micro-servicio para identificar la siguiente acción a realizar. En el caso de que la respuesta sea positiva, se recibe la palabra "OK" y, en caso contrario, la palabra "NOT". Esto indica si se ha detectado o no la palabra de activación.

Cuando se recibe una respuesta negativa por parte del micro-servicio, la tarea `PostTask` continua con el envío de los elementos que se encuentren dentro de la Cola de peticiones hasta recibir una respuesta positiva. Estos elementos dentro de la Cola de peticiones son los `buffers` de un segundo, grabados con el fin de identificar la palabra de activación. Por esta razón, mientras no sea identificada dicha palabra, se tiene un envío constante de `buffers` de audio de un segundo al micro-servicio, mediante la tarea `PostTask`.

Cuando se detecta la palabra de activación y la respuesta del micro-servicio es positiva, las acciones a realizar son las siguientes: 1) se eliminan todos los elementos contenidos dentro de la Cola de peticiones, ya que estos elementos son `buffers` que contienen la información de un segundo de audio que ya no es requerido, debido a que se ha identificado la palabra de activación y se libera la memoria utilizada, permitiendo utilizarla para la grabación de la petición del usuario; y 2) se detiene la tarea `WakeWordTask` mediante la función `vTaskSuspend`, se activa la tarea `AudioRecorderTask`, que a su vez activa la tarea `LigtherTask` y se cambia el valor de la bandera a `false`. Así la siguiente vez que la tarea `PostTask` detecte que existe un elemento dentro de la cola de peticiones, sabrá que es un elemento que deberá ser enviado a otro `endpoint` del micro-servicio, encargado del procesamiento de la petición del usuario.

En caso de que el micro-servicio no logre recibir la petición, se muestra en la pantalla un mensaje de *HTTP request failed* y se prosigue al envío del siguiente elemento contenido en la cola de peticiones. La Figura 4.24 muestra un claro ejemplo del funcionamiento del sistema de captura permanente mediante mensajes mostrados en la pantalla, indicando que el proceso se ha llevado a cabo.

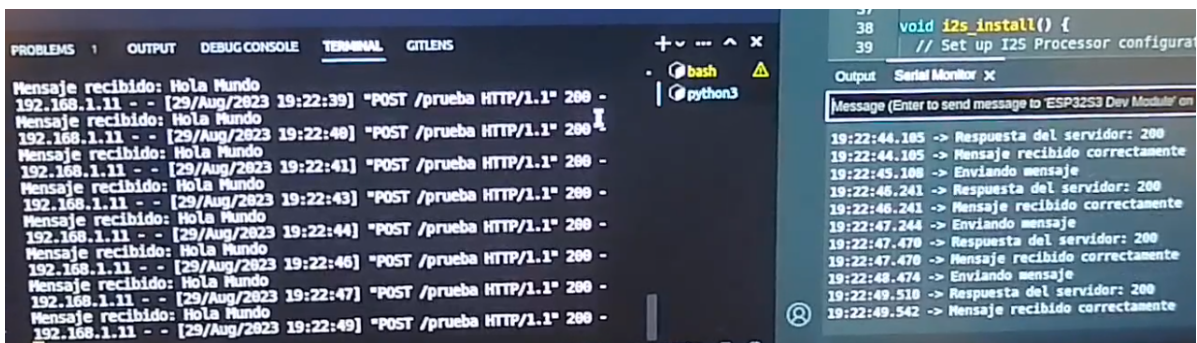
```

14:56:35.445 -> Connecting to WiFi...
14:56:35.445 -> Connected to WiFi
14:56:55.929 -> Buffer 1 creado
14:56:55.929 -> Sending message
14:57:00.939 -> HTTP request failed
14:57:00.939 -> -1
14:57:04.850 -> Buffer 2 creado
14:57:04.850 -> Sending message
14:57:05.204 -> PasoRecording user request...
14:57:10.178 -> User request recorded

```

Figura 4.24: Captura de petición de usuario mediante sistema de captura permanente

Así mismo, en la Figura 4.25 se muestra la comunicación entre el micro-servicio y el sistema de captura permanente para el envío constante de peticiones al micro-servicio.



```

Mensaje recibido: Hola Mundo
192.168.1.11 - - [29/Aug/2023 19:22:39] "POST /prueba HTTP/1.1" 200 -
Mensaje recibido: Hola Mundo
192.168.1.11 - - [29/Aug/2023 19:22:40] "POST /prueba HTTP/1.1" 200 -
Mensaje recibido: Hola Mundo
192.168.1.11 - - [29/Aug/2023 19:22:41] "POST /prueba HTTP/1.1" 200 -
Mensaje recibido: Hola Mundo
192.168.1.11 - - [29/Aug/2023 19:22:43] "POST /prueba HTTP/1.1" 200 -
Mensaje recibido: Hola Mundo
192.168.1.11 - - [29/Aug/2023 19:22:44] "POST /prueba HTTP/1.1" 200 -
Mensaje recibido: Hola Mundo
192.168.1.11 - - [29/Aug/2023 19:22:46] "POST /prueba HTTP/1.1" 200 -
Mensaje recibido: Hola Mundo
192.168.1.11 - - [29/Aug/2023 19:22:47] "POST /prueba HTTP/1.1" 200 -
Mensaje recibido: Hola Mundo
192.168.1.11 - - [29/Aug/2023 19:22:49] "POST /prueba HTTP/1.1" 200 -

void i2s_install() {
38 // Set up I2S Processor configura
39
Output Serial Monitor X
Message (Enter to send message to 'ESP32S3 Dev Module' on
19:22:44.105 -> Respuesta del servidor: 200
19:22:44.105 -> Mensaje recibido correctamente
19:22:45.108 -> Enviando mensaje
19:22:46.241 -> Respuesta del servidor: 200
19:22:46.241 -> Mensaje recibido correctamente
19:22:47.244 -> Enviando mensaje
19:22:47.470 -> Respuesta del servidor: 200
19:22:47.470 -> Mensaje recibido correctamente
19:22:48.474 -> Enviando mensaje
19:22:49.510 -> Respuesta del servidor: 200
19:22:49.542 -> Mensaje recibido correctamente

```

Figura 4.25: Comunicación entre el sistema de captura permanente y el micro-servicio

4.2.4. Palabra de activación

Para la selección de la palabra de activación, se tomaron en consideración cinco palabras cortas, las cuales se sometieron 20 veces cada una a evaluaciones de tiempo de traducción de audio a texto, así como el porcentaje de las veces que la palabra de activación fue traducida e identificada correctamente por el *framework* *Whisper*.

Las palabras seleccionadas se pueden apreciar en la Tabla 4.3. Como se puede observar, se tomaron en consideración tres nombres propios (Alexa, Marvin e Iván) de los cuales sólo uno (Alexa) fue traducido en un tiempo menor a dos segundos, sin embargo, el porcentaje de traducción correcta de la palabra fue menor al 50%. Por otra parte, los resultados obtenidos al utilizar los nombres Marvin e Iván fueron deficientes, ya que el

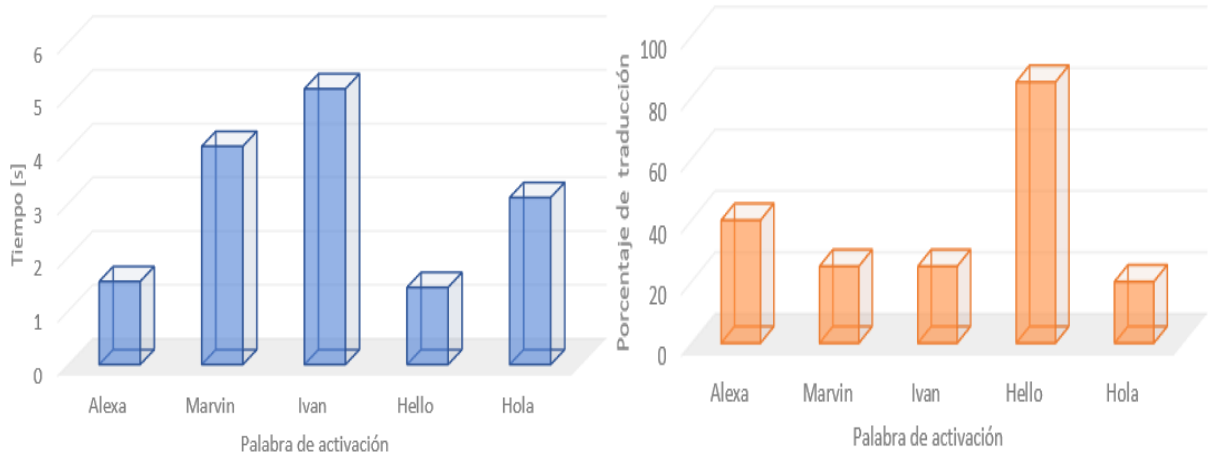
tiempo de traducción superó los tres segundos y el porcentaje de traducción correcta fue menor al 30 %. Estos resultados se atribuyen a la falta de información proporcionada al *framework Whisper*, ya que la traducción de una sola palabra dificulta la identificación del idioma utilizado y, posteriormente, la detección del nombre propio. Sin embargo, si se proporciona más información a *Whisper*, como una frase en la que se menciona un nombre propio, en medio o al final de la oración, el tiempo de traducción disminuye y el porcentaje de traducción correcta aumenta.

Tabla 4.3: Comparativa de evaluación entre palabras de activación

Palabra de activación	Tiempo de traducción [s]	Porcentaje de aciertos en traducción
Alexa	1.54	40 %
Marvin	4.06	25 %
Ivan	5.13	25 %
Hello	1.43	85 %
Hola	3.11	20 %

Por otro lado, como se muestra en la Tabla 4.3 y en las Figuras 4.26a y 4.26b, las otras palabras consideradas, basadas en la información del párrafo anterior (*Hello* y *Hola*), a pesar de tener el mismo significado, difieren significativamente en el tiempo de traducción, debido a que se encuentran en diferentes idiomas. La palabra “*Hello*” es la más rápida en ser traducida, con un promedio de 1.43 segundos, mientras que la palabra “*Hola*” tarda un poco más de tres segundos y tiene un porcentaje de traducción correcta por debajo del 50 %, en tanto que la palabra “*Hello*” alcanzó un 85 % de porcentaje de traducción. Estas diferencias en los resultados entre las dos palabras mencionadas se deben también al hecho de que *Whisper* tiende actualmente a proporcionar mejores traducciones para el audio en inglés que en español, gracias a que el conjunto de datos en inglés contiene más información que el de español y, en consecuencia, un mejor entrenamiento.

Finalmente, con base en la información recopilada después de realizar las comparaciones, se tomó la decisión de utilizar la palabra *Hello* como palabra de activación del sistema de captura permanente, utilizando el dispositivo ESP32-S3.



(a) Tiempo promedio de traducción de la palabra de activación

(b) Porcentaje de traducciones correctas de palabras de activación

4.2.5. Rango de captura de audio del sistema de captura permanente

Se realizaron pruebas para la evaluación del rango de captura de audio del sistema de captura permanente, con el fin de determinar la distancia en la cual el sistema es capaz de identificar la palabra de activación y capturar la petición del usuario de manera efectiva. Las pruebas se realizaron a distancias de 1, 2, 3 y 4 metros entre el usuario y el sistema de captura permanente. Se realizaron un total de 20 pruebas por cada distancia mencionada.

Tabla 4.4: Pruebas de identificación de la palabra de activación a distintas distancias

Distancia [m]	Precisión de detección
1	90 %
2	75 %
3	25 %
4	10 %

Con base en los resultados presentados en la Tabla 4.4 y en la Figura 4.27, se puede observar que si se considera un porcentaje aceptable igual o mayor al 75 %, lo que significa que para cada 20 solicitudes, el sistema de captura permanente logra identificar un mínimo de 15, es posible utilizarlo desde una distancia de dos metros. Esto es razonable, considerando que el sistema de reconocimiento de voz y de habla está diseñado para tener varios sistemas de captura permanente, posicionados estratégicamente, y también se dispone de una aplicación Android (ver Sección 4.1) que permite al usuario seguir realizando peticiones al micro-servicio desde cualquier lugar dentro del alcance de la señal de la red.

Por otro lado, si el usuario se encuentra a una distancia entre uno y uno y medio me-

tros, el porcentaje aumenta a entre el 80 % y el 90 %. Finalmente, se puede afirmar que, a distancias superiores a 2 metros entre el usuario y el dispositivo de captura permanente, el sistema de captura permanente tiende a no identificar correctamente las solicitudes del usuario o a veces no reconoce cuándo se menciona la palabra clave. Por lo tanto, en tales escenarios se requiere la colocación de otro dispositivo de captura permanente que permita capturar correctamente las solicitudes del usuario.

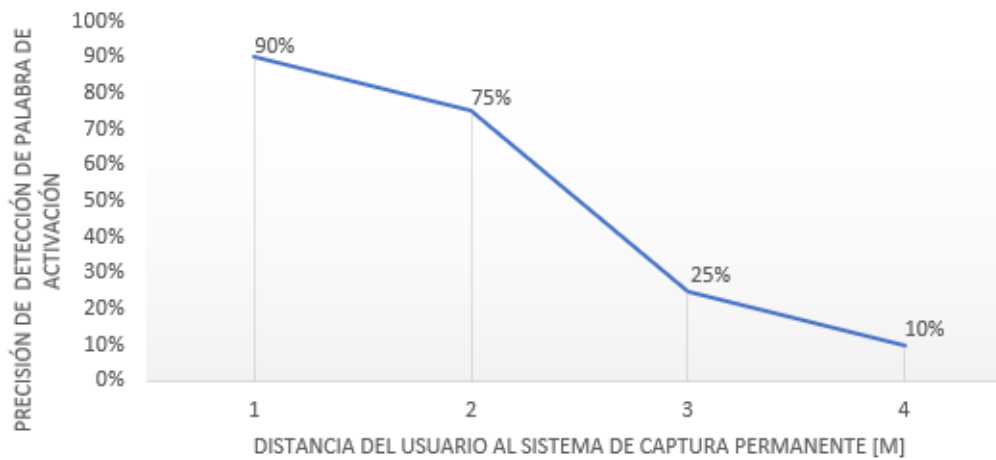


Figura 4.27: Porcentaje de precisión en la detección de la palabra de activación a diferentes distancias.

4.3. Micro-Servicio

A continuación, se presenta la implementación del micro-servicio desarrollado para el sistema de reconocimiento del voz y habla multi-fuente orientado a compañeros digitales. En la Sección 4.3.1 se justifica la elección la tarjeta de desarrollo a utilizar, seguida de la Sección 4.3.2 en donde se abordan los *frameworks* utilizados para la implementación del micro-servicio. En la Sección 4.3.3 se describe la inicialización del micro-servicio y en la Sección 4.3.4 se aborda la generación de los repositorios utilizados. Posteriormente, en la Sección 4.3.5 se explica el manejo de los archivos de audio, seguida de la Sección 4.3.6 donde se expone la generación del modelo para la identificación de usuarios. A continuación, en la Sección 4.3.7 se detalla el proceso de extracción de información relevante de la petición del usuario y finalmente, en la Sección 4.3.8, se aborda el envío de comandos a los sistemas externos.

4.3.1. Selección de dispositivo para el micro-servicio

Para la selección del dispositivo apropiado a utilizar para la implementación del micro-servicio se realizó una comparativa de las características de interés de algunos compañeros digitales que se encuentran en el mercado. Los dispositivos son: *Google Home mini*, *Google Home*, *Echo Dot 4ta* y *Echo Dot 5ta*. Las características tomadas en consideración son: 1) sistema operativo (SO), 2) procesador y 3) conectividad.

Como se puede observar en la Tabla 4.5, dentro de las características que comparten estos compañeros digitales, son el sistema operativo, la utilización de procesadores que cuentan con más de un núcleo y la conectividad mediante Wi-Fi y Bluetooth 5.0. En el caso de los compañeros digitales *Echo Dot 4a* y *5a*, estos cuentan con un procesador con menor cantidad de núcleos, pero con una mayor frecuencia de trabajo y cuentan con un módulo independiente que mejora la velocidad de procesamiento de lenguaje natural (*AZ1 Neural Edge* y *AZ2 Neural Edge*, respectivamente).

Tabla 4.5: Comparativa de características de VIPAs actuales en el mercado

Asistente	Sistema Operativo	Procesador	Conectividad
<i>Google Home Mini</i>	Android IOS	<ul style="list-style-type: none"> ■ AMR de 64 bits y cuatro núcleos con 1.4Ghz 	<ul style="list-style-type: none"> ■ Wi-fi 802.11b/g/n/ac (2.4 GHz/5 GHz) ■ Bluetooth 5.0
<i>Google Home</i>	Android IOS	<ul style="list-style-type: none"> ■ A53 de cuatro núcleos a 1.8GHz 	<ul style="list-style-type: none"> ■ Wi-fi 802.11b/g/n/ac (2.4 GHz/5 GHz) ■ Bluetooth 5.0
<i>Echo Dot 4ta</i>	Fire OS Android IOS	<ul style="list-style-type: none"> ■ MediaTek de dos núcleos a 2GHz ■ AZ1 Neural Edge ■ SDRAM 4GB 	<ul style="list-style-type: none"> ■ Wi-fi 802.11b/g/n/ac (2.4 GHz/5 GHz) ■ Bluetooth 5.0
<i>Echo Dot 5ta</i>	Fire OS Android IOS	<ul style="list-style-type: none"> ■ MediaTek de dos núcleos a 2GHz ■ AZ2 Neural Edge ■ SDRAM 4GB 	<ul style="list-style-type: none"> ■ Wi-fi 802.11b/g/n/ac (2.4 GHz/5 GHz) ■ Bluetooth 5.0

Se realizó una búsqueda de tarjetas de desarrollo con características semejantes a los compañeros digitales evaluados, que permitieran llevar a cabo la implementación de los módulos que conforman el micro-servicio. Se seleccionaron tres tarjetas de desarrollo y se realizó una comparativa de sus características, tomando en consideración las características de los compañeros digitales evaluados. Las tarjetas seleccionadas y sus características se pueden visualizar en la Tabla 4.6.

Como se puede observar en la Tabla 4.6, estas tres tarjetas de desarrollo cuentan un sistema operativo que funciona bajo una distribución Linux y tienen un procesador Cortex de cuatro núcleos, pero de diferentes versiones, siendo la tarjeta *Raspberry Pi 4B* la que cuenta con una versión más moderna con el procesador Cortex-A72. Sin embargo, uno de las características de importancia es la memoria RAM con la que cuentan, por lo cual debido a la baja capacidad de memoria RAM de la tarjeta *RaspBerry Pi 3B+*, queda descartada para su utilización.

Tabla 4.6: Comparativa de características de tarjetas de desarrollo consideradas para implementación del reconocedor de voz y habla

Tarjeta de desarrollo	CPU	GPU	Memoria RAM	Memoria ROM	Conectividad
<i>Raspberry Pi 3B+</i>	Cortex-A53 de cuatro núcleos a 1.4 GHz	No	1 GB	Tarjeta Flah de 8 a 32 GB	<ul style="list-style-type: none"> ■ Wi-Fi 802.11n ■ Bluetooth 4.2
<i>Raspberry 4B</i>	Cortex-A72 de cuatro núcleos a 1.5 GHz	No	2/4/8 GB	Tarjeta Flash de 8 a 32 GB	<ul style="list-style-type: none"> ■ Wi-Fi 802.11ac. ■ Bluetooth 5.0
<i>Jetson Nano</i>	Cortex-A57 de cuatro núcleos a 1.5 GHz	Arquitectura NVIDIA Maxwell™ con 128 núcleos NVIDIA CUDA	4 GB	Tarjeta Flah eMMC5.1 de 16 GB	<ul style="list-style-type: none"> ■ No Wi-Fi ■ No Bluetooth

Finalmente, se decidió utilizar la tarjeta de desarrollo *RaspBerry Pi 4B*, debido a que a pesar de que la tarjeta *Jetson Nano* cuenta con una GPU que puede ayudar a mejorar la velocidad de procesamiento y tiene una memoria RAM similar a la de los compañeros digitales analizados, no cuenta con módulos de comunicación inalámbrica para Wi-Fi ni Bluetooth, lo cual es uno de los factores más importantes para el desarrollo de este

proyecto, debido a la necesidad de comunicación con diversos dispositivos.

4.3.2. Frameworks

A continuación se abordan los análisis de los *frameworks* utilizados para la implementación del micro-servicio. En la Subsección 4.3.2.1 se habla sobre el *framework* `Whisper`, seguido de la Subsección 4.3.2.2 en donde se describe `OpenVINO`. Posteriormente, se detalla en la Subsección 4.3.2.3 el *framework* `spaCy` para la obtención de características y finalmente en la Subsección 4.3.2.4 se expone el *framework* `Kafka`.

4.3.2.1. Whisper

Para la traducción de los archivos de audio recibidos a un formato de texto se utiliza el *framework* `Whisper`, el cual es un sistema de reconocimiento del habla automático que utiliza, para el procesamiento de lenguaje natural, un modelo denominado *encoder-decoder Transformer* entrenado con 680,000 horas de datos supervisados plurilingües y multitarea recopilados de la web [Radford et al., 2023].

Se decidió utilizar este *framework* debido a que cuenta con las siguientes características:

- Permite la traducción de los archivos de audio a texto en distintos idiomas.
- Cuenta con diferentes modelos para la traducción de los archivos de audio a texto que varían en su robustez, implicando que para mayor eficiencia en la traducción, mayor es el tiempo de procesamiento y viceversa.
- Permite la implementación de los modelos de manera local, lo que evita el uso de cómputo en la nube y la compartición de los datos de los usuarios.
- Tiene la capacidad de ser implementado de manera multi-hilo e, incluso, de manera paralela, utilizando una GPU para ayudar a la aceleración de su ejecución.

Para determinar el tiempo de traducción requerido y la precisión de traducción de los diferentes modelos (`Tiny`, `Base`, `Small`, `Medium`) con los que cuenta este *framework*, se elaboraron pruebas de precisión y velocidad para la traducción de audio a texto. Los datos de prueba que se utilizaron fueron archivos de audio que van desde los 10 hasta los 55 segundos en incrementos de cinco segundos. Las pruebas fueron realizadas, utilizando el mayor número de hilos disponibles en el dispositivo.

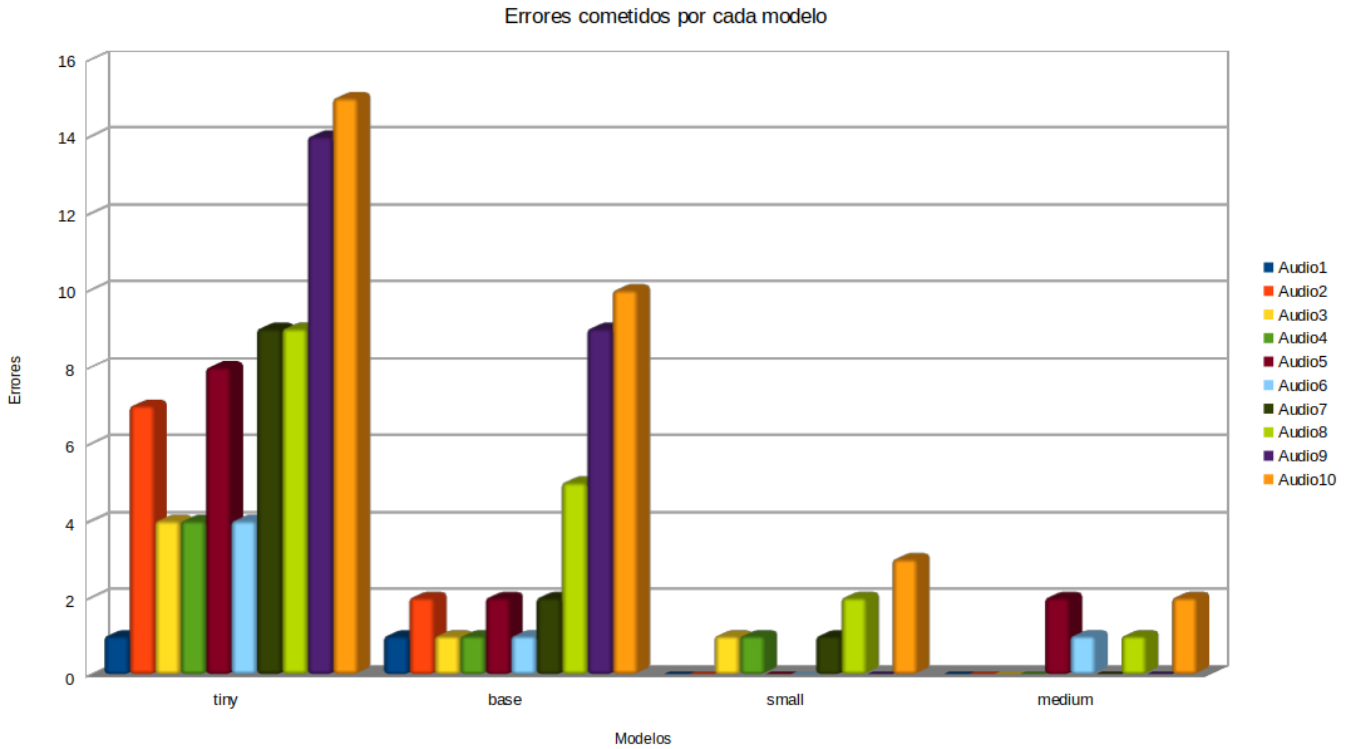


Figura 4.28: Resultados obtenidos en pruebas de traducción, utilizando los diferentes modelos de *Whisper*

Como se puede observar en la Figura 4.28, dependiendo del modelo la precisión varía la traducción de los archivos de audio, siendo el modelo *Tiny* el que más errores tiende a cometer y el modelo *Medium* el más preciso. Sin embargo, como se puede observar en la Figura 4.29, los tiempos de traducción de los modelos *Tiny* y *Medium* varían considerablemente, ya que para audios de cinco segundos (prueba 1), el tiempo de traducción es de aproximadamente uno y cinco segundos, respectivamente, teniendo cuatro segundos de diferencia. Así mismo, se observa que mientras mayor sea el tiempo del audio a traducir, el tiempo que incrementa en ambos modelos no es proporcional, ya que para el caso de un audio de 55 segundos (prueba 10) el tiempo de traducción que toma el modelo *Tiny* es de aproximadamente cinco segundos, mientras que para el modelo *Medium* el tiempo de traducción se incrementa a aproximadamente 45 segundos, existiendo una diferencia entre ambos modelos de 40 segundos.

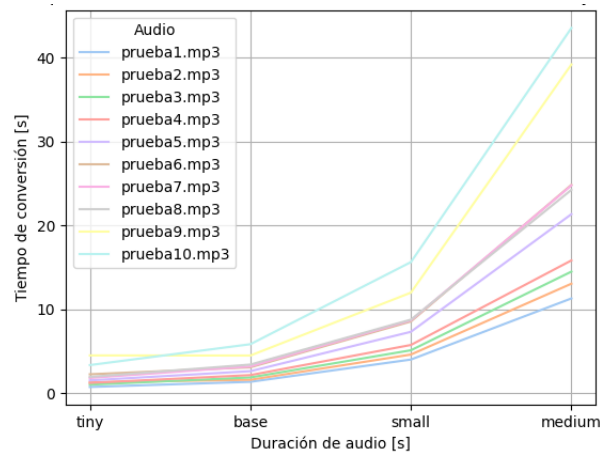


Figura 4.29: Resultados obtenidos en pruebas de tiempo de traducción del *framework* *Whisper* utilizando una *laptop* y cuatro hilos

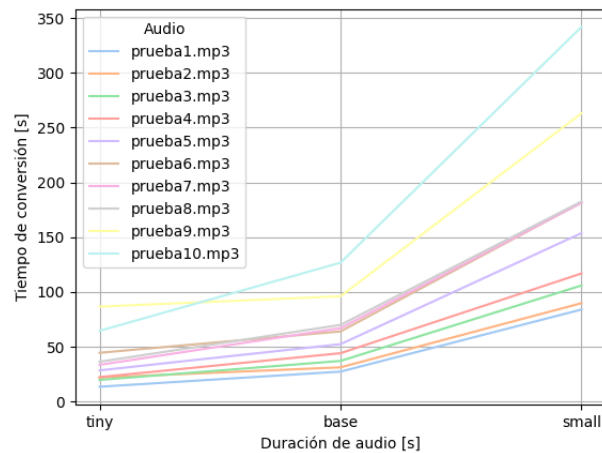


Figura 4.30: Resultados obtenidos en pruebas de tiempo de traducción del *framework* *Whisper* utilizando la tarjeta *RaspBerry Pi 4B* y cuatro hilos

Los resultados derivados de las pruebas de evaluación del tiempo de traducción de *Whisper* en la tarjeta de desarrollo *RaspBerry Pi 4B* señalan que la viabilidad de implementar este sistema en dicha plataforma es cuestionable. A pesar de haber utilizado el mayor número posible de hilos, los resultados para audios de cinco segundos (prueba 1) superan los 10 segundos, inclusive en el modelo *Tiny*. Como se ilustra en la Figura 4.30, el tiempo de traducción fue aproximadamente de 15 segundos para el modelo *Tiny*, mientras que para el modelo *Small*, el tiempo se eleva a más de 50 segundos. Además, en el intento de implementar el modelo *Medium*, la tarjeta de desarrollo no fue capaz de gestionarlo.

Considerando que se busca que el proceso de captura de solicitudes de usuarios tenga un tiempo aproximado entre 10 y 15 segundos (pruebas 2 y 3, respectivamente), se proyecta que la traducción utilizando el modelo `Tiny` requeriría aproximadamente entre 25 y 27 segundos. Es importante destacar que la precisión de la traducción de archivos de audio con este modelo sería deficiente. Por otro lado, si se utiliza el modelo `Small`, que ofrece una mayor precisión en la traducción, el tiempo estimado de traducción para una petición de 10 segundos sería alrededor de 90 segundos, y para una petición de 15 segundos, superaría los 100 segundos (1.40 minutos), lo cual resulta inviable en términos de eficiencia y tiempo de respuesta para el usuario.

Con base en los resultados obtenidos y las consideraciones previas, se ha tomado la decisión de no implementar el micro-servicio en la tarjeta *Raspberry Pi 4B* propuesta. En su lugar, se optará por la implementación del micro-servicio en la *laptop* utilizada para la comparativa de tiempos de traducción. Aunque los tiempos de traducción aún no cumplen con las expectativas deseadas, se exploró la posibilidad de utilizar herramientas como `OpenVino` para acelerar el proceso de traducción en el entorno de la *laptop*. Esta elección se fundamenta en la búsqueda de un equilibrio entre la eficiencia del sistema y la capacidad de optimización a través de herramientas específicas.

4.3.2.2. OpenVINO

`OpenVINO` proporciona un conjunto de herramientas y bibliotecas desarrolladas por Intel para optimizar y acelerar la ejecución de modelos de redes neuronales en una variedad de dispositivos, desde sistemas integrados hasta servidores de alto rendimiento. Su principal objetivo es ofrecer un rendimiento eficiente y maximizar la velocidad de inferencia de modelos de aprendizaje profundo.

La aceleración proporcionada por `OpenVINO` se basa en la optimización específica para la arquitectura del hardware subyacente. Utiliza técnicas avanzadas de compilación y optimización para adaptar los modelos de redes neuronales al hardware específico, aprovechando características como la presencia de unidades de procesamiento especializadas para mejorar significativamente el rendimiento.

En la actualidad, `OpenVINO` ofrece proyectos que destacan en la optimización de modelos de aprendizaje profundo en diversas áreas, abordando, e.g., la detección de objetos mediante redes convolucionales como YOLO y, más recientemente, la optimización en redes de modelos de lenguaje grande (LLM). En el caso específico de `Whisper`, se dispone de un proyecto diseñado para su mejora¹.

¹<https://docs.openvino.ai/2023.1/notebooks/227-whisper-subtitles-generation-with-output.html#convert-whisper-decoder-to-openvino-ir>

Aunque el proyecto proporcionado por la página oficial de **OpenVINO** está inicialmente orientado a optimizar la traducción de audios de vídeos en **YouTube**, los principios de optimización son aplicables en el sistema desarrollado. Se ha tomado la decisión de emplear este proyecto de optimización de **Whisper**, debido a los resultados reportados en la Sección 4.3.2.1 y a que el dispositivo donde se desarrolla el micro-servicio cuenta con un procesador Intel, lo cual permite la implementación eficiente de **OpenVINO**.

La optimización consiste en sustituir el modelo *decoder-encoder* en el cual se basa **Whisper**, a un formato **OpenVINO Intermediary Representation (IR)**, el cual es una representación intermedia de un modelo de red neuronal entrenado. El formato IR proporciona una representación independiente de la plataforma y del *framework* original del modelo, lo que permite la optimización y la ejecución eficiente en diferentes dispositivos de *hardware*. **OpenVINO** utiliza el formato IR para convertir modelos entrenados en diferentes *frameworks*, como **TensorFlow** o **PyTorch**, a un formato optimizado que puede ser implementado en *hardware* específico, como CPUs, GPUs, etc.

Al sustituir el modelo *decoder-encoder* original de **Whisper** por el modelo optimizado de **OpenVINO**, se volvieron a realizar pruebas de eficiencia con los mismos archivos de audio que se utilizaron para las primeras pruebas reportadas en la Sección 4.3.2.1. En esta ocasión, se realizó una comparativa entre los tiempos de traducción utilizando cuatro hilos y los tiempos de traducción utilizando el modelo optimizado por **OpenVINO** para el modelo **Medium**, los cuales se muestran en la Figura 4.31.

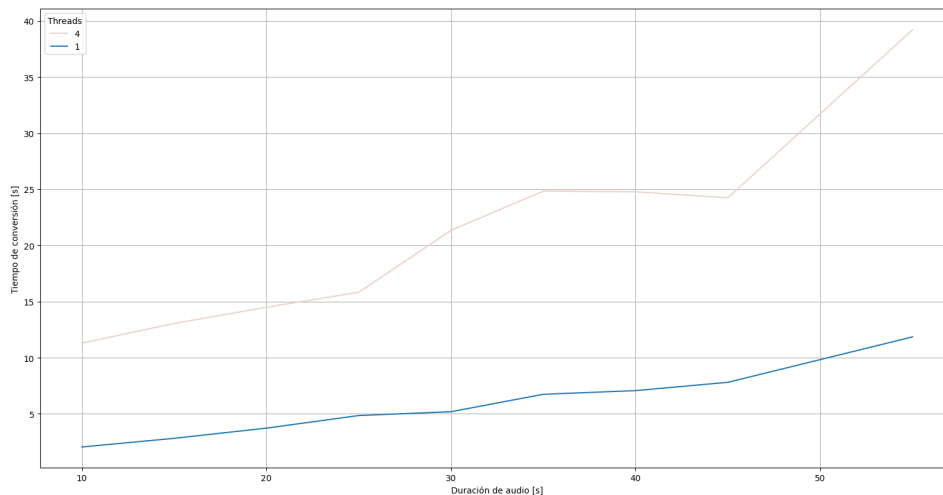


Figura 4.31: Comparativa de tiempos de traducción de archivos de audio utilizando **Whisper** con una implementación de cuatro hilos y con **OpenVINO**.

Los resultados obtenidos revelan que la integración de **OpenVINO** en modelos más robustos, como el modelo **Medium**, que ofrece una mayor precisión en la traducción de archivos

de audio, conduce a resultados altamente positivos. La Figura 4.31 ilustra claramente cómo los tiempos de traducción se reducen, de manera significativa, al emplear `OpenVINO` en comparación con la implementación multi-hilo. Por ejemplo, al traducir archivos de 10 segundos utilizando el modelo `Medium` con la implementación multi-hilo, el tiempo es aproximadamente de 12 segundos. En contraste, al utilizar `OpenVino`, el tiempo de traducción se reduce a aproximadamente dos segundos, indicando una traducción seis veces más rápida.

Estos resultados respaldan la viabilidad de implementar el micro-servicio en una laptop, especialmente cuando se aprovecha `OpenVINO`. La capacidad de obtener traducciones de audio a texto en tiempos considerablemente reducidos hace que esta elección sea práctica y eficaz para aplicaciones en tiempo real. La optimización proporcionada por `OpenVINO` demuestra ser una estrategia efectiva para mejorar la eficiencia del sistema y garantizar traducciones rápidas y precisas.

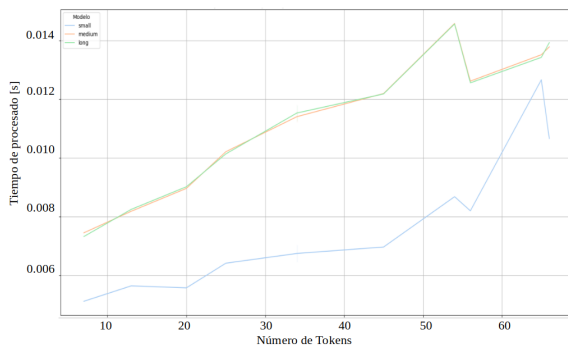
4.3.2.3. SpaCy

Para la obtención de la información relevante y requerida para la elaboración del formato *json*, que permite la comunicación con los sistemas externos, se decidió la utilización del *framework* `spaCy` el cual, al igual que `Whisper`, es una herramienta que permite el procesamiento de lenguaje natural, utilizando el lenguaje de programación en Python [Honnibal et al., 2020b]. Las características por las cuales se decidió utilizar este *framework* son:

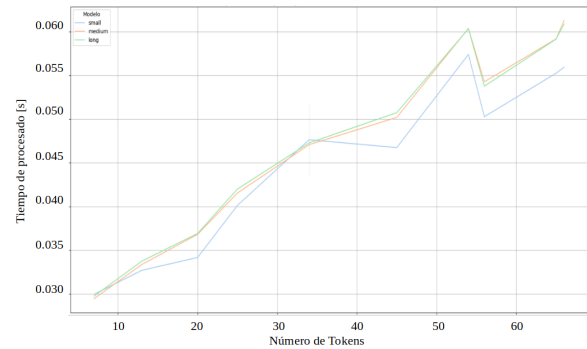
- Es un *framework* de código abierto diseñado para la producción de código de manera eficiente.
- Destaca por su velocidad en el procesamiento de texto y es capaz de procesar grandes cantidades de texto.
- Ofrece funciones de procesamiento de lenguaje natural de extremo a extremo, incluyendo tokenización, lematización, análisis gramatical, reconocimiento de entidades nombradas (NER), etc.
- Al igual que `Whisper`, `spaCy` incluye modelos preentrenados, utilizando grandes cantidades de datos para la elaboración de diversas tareas de procesamiento de lenguaje natural.
- Es de fácil integración con proyectos desarrollados en el lenguaje de programación Python.

Al igual que `Whisper`, los modelos con los que cuenta `spaCy` varían en su eficiencia y velocidad de procesamiento. Mientras más rápido sea el modelo, menor es la cantidad

de funciones que este puede realizar. Se elaboraron pruebas de velocidad de los modelos, utilizando archivos de texto con diferente cantidad de palabras, comenzando desde cinco hasta 65 palabras. Las pruebas fueron realizadas en dos dispositivos. El primero es una *laptop* con sistema operativo Ubuntu, procesador Intel Core i5 9th gen, 4 núcleos y 8 GB de memoria RAM. El segundo dispositivo es la tarjeta RaspBerry Pi 4B, la cual cuenta con las características mencionadas en la Tabla 4.6 de la sección 4.3.1.



(a) Análisis de tiempo de modelos de spaCy preentrenados en una *laptop*



(b) Análisis de tiempo de modelos de spaCy preentrenados en una RaspBerry Pi 4B

Figura 4.32: Comparativa de pruebas de velocidad de modelos de spaCy en dos dispositivos diferentes

Como se puede observar en las Figuras 4.32a y 4.32b, ninguno de los modelos implementados en ambos dispositivos supera un tiempo de procesamiento de seis micro segundos. En el caso de la *laptop*, el tiempo máximo corresponde a poco más de 14 microsegundos, mientras que en la RaspBerry Pi el tiempo máximo no supera los 60 microsegundos. Estos resultados indican que se puede utilizar, sin problema, el modelo más robusto de spaCy para el módulo de procesamiento de lenguaje natural del micro-servicio y esperar resultados en menos de un segundo. Esto le permite ser implementado tanto en dispositivos con grandes prestaciones como es el caso de una *laptop*, como en dispositivos de menores prestaciones como la tarjeta *RaspBerry Pi*.

4.3.2.4. Kafka

Para la selección del *framework* a utilizar para el envío de mensajes mediante el concepto de productor-consumidor se realizó una comparativa entre dos herramientas de Apache, las cuales son Kafka y ActiveMQ.

Tabla 4.7: Comparación entre Apache Kafka y Apache ActiveMQ

Kafka	ActiveMQ
Modelo de registro de eventos (<i>topics</i>), <i>Publisher - Suscriber</i>	Modelo de cola de mensaje, <i>Producer - Consumer</i>
Escalable, distribuido, permite partición de datos y es tolerante a fallos	Escalable pero menos eficiente en procesamiento de grandes volúmenes
Almacenamiento de mensajes en un registro de eventos inmutable y persistente	Almacenamiento de datos en disco de forma predeterminada
Útil para manejo de datos en tiempo real	Útil para manejo de datos en orden
Útil para la construcción de arquitecturas de micro-servicio	Ideal para la integración en arquitecturas de microservicios
Soporte de particionamiento y replicación para alta disponibilidad y tolerancia a fallos	Soporte de replicación y <i>clustering</i> para alta disponibilidad

La Tabla 4.7 presenta una comparativa entre ambos *frameworks*, revelando que ambos cumplen con características esenciales para ser implementados en este micro-servicio. A pesar de su idoneidad general para micro-servicios, **Kafka** destaca al satisfacer un requisito adicional crucial para el correcto funcionamiento del micro-servicio: su capacidad para el manejo de datos en tiempo real. Esto se debe a su arquitectura basada en registro de eventos, que permite el procesamiento inmediato de flujos continuos de datos. En contraste, **ActiveMQ**, al basarse en una arquitectura de encolamiento, es especialmente útil para el manejo de datos de forma ordenada, pero su enfoque puede no ser tan eficiente para situaciones que requieren procesamiento inmediato.

La preferencia por **Kafka** se basa en su escalabilidad, distribución, capacidad para la partición de datos, tolerancia a fallos y su utilidad para la construcción de arquitecturas de micro-servicios. Además, su soporte de particionamiento y replicación para alta disponibilidad y tolerancia a fallos lo convierte en una elección sólida para un entorno dinámico como el micro-servicio en consideración.

En resumen, la elección de **Kafka** sobre **ActiveMQ** se justifica por la necesidad de manejar datos en tiempo real y por las características específicas que ofrece **Kafka** para entornos distribuidos y orientados a micro-servicios.

4.3.3. Inicialización del micro-servicio

La puesta en marcha del micro-servicio implica la inicialización de sus tres componentes principales: el módulo de identificación de usuarios, el módulo de procesamiento de lenguaje natural y el módulo de comunicación. Para llevar a cabo la inicialización de estos módulos, se hace uso de dos archivos de configuración que contienen toda la información necesaria para el correcto funcionamiento del micro-servicio.

El primer archivo, denominado `MainConfig.json`, contiene la información relacionada con la configuración esencial para utilizar los *frameworks* requeridos, así como los nombres de dos archivos adicionales que se utilizan para configurar el micro-servicio, según el idioma seleccionado. Como se puede observar en la Figura 4.33, para el módulo de identificación de usuarios, el archivo de configuración incluye el nombre del modelo preentrenado a utilizar para el reconocimiento de los usuarios. En cuanto al módulo de procesamiento de lenguaje natural, este archivo contiene la configuración necesaria para la inicialización de los *frameworks* `Whisper` y `Spacy`, los cuales son esenciales para identificar el tipo de solicitud y extraer la información necesaria. Cabe mencionar que el archivo de configuración no contiene información relacionada con el módulo de comunicación, ya que dicha información está contenida en los archivos `EnglishConfig.json` y `SpanishConfig.json`.



```
{
  "WHISPER": {
    "whisper_model": "medium",
    "device": "cpu",
    "NEW_CONFIG": {
      "encoder": "whisper_encoder.xml",
      "decoder": "whisper_decoder.xml"
    }
  },
  "SPACY": {
    "MODELS": {
      "spanish": "es_core_news_md",
      "english": "en_core_web_lg"
    }
  },
  "CONFIG": {
    "english": "EnglishConfig.json",
    "spanish": "SpanishConfig.json"
  },
  "MODEL": {
    "model": "voice-identification-model.h5"
  }
}
```

Figura 4.33: Archivo de configuración `MainConfig.json`

Dentro de la configuración de `Whisper` se ha seleccionado el modelo `Medium` ejecutándose en una CPU. Así mismo, se incluyen los nombres de dos archivos previamente creados (`Whisper_encoder.xml` y `whisper_decoder.xml`). Estos archivos son esenciales, ya que

reemplazan a los archivos originales de *Whisper*, permitiendo su implementación en dispositivos compatibles con el *framework* *OpenVino* y mejorando significativamente el tiempo de traducción de archivos de audio.

Por otra parte, para la inicialización de *spaCy*, el archivo de configuración contiene los nombres de los modelos a utilizar, dependiendo del idioma que se haya seleccionado, siendo el modelo `es_core_news_md` para el idioma español y `en_core_web_lg` para el idioma en inglés. Además, se puede observar que este archivo de configuración también contiene el nombre de los archivos utilizados para la configuración del idioma que se desea utilizar en el micro-servicio (`EnglishConfig.json` y `SpanishConfig.json`). Estos tres archivos son utilizados posteriormente por las clases `MainConfiguration` y `Language`.

Los archivos de configuración para los idiomas inglés y español contienen todos los elementos esenciales para la identificación y procesamiento de peticiones realizadas por el usuario. Además, incluyen los componentes necesarios para el envío de estas peticiones a través del módulo de comunicación, adaptándose a casos de estudio específicos que pueden abordarse en el idioma seleccionado.

Dentro de la información almacenada en los archivos de configuración del idioma, se encuentra la estructura del formato *json* esencial para la comunicación con sistemas externos, como el formato *json* diseñado para interactuar con el *broker* de un compañero digital. Así mismo, se detallan los *endpoints* utilizados en el módulo de comunicación para transmitir el formato *json* correspondiente. Las Figuras 4.34a y 4.34b presentan ejemplos concretos de los formatos *json* elaborados y contenidos en el archivo de configuración `EnglishConfig.json`, junto con algunos *endpoints* empleados para la interacción con dispositivos del Internet de las Cosas.

```

{} EnglishConfig.json x
fig.json > {} INSTRUCTIONS > {} robot >
"INSTRUCTIONS":{
  "lamp":{
    "on":"","
    "color":"","
    "brightness":""
  },
  "blinds":{
    "state":""
  },
}

```

(a) Formatos *json* utilizados para la interacción con sistemas externos

```

sh:Config.json > {} ENDPOINTS > {} robot
ENDPOINTS:{
  "lamp":"http://10.2.2.33:1880/r400/kimLamp",
  "blinds":"http://10.2.2.1880/knx/floor_1/room_102/blinds",
  "robot":{
    "get":{
      "operator": "https://api.interactions.ics.unisg.ch/cherrybot/operator",
      "tcp": "https://api.interactions.ics.unisg.ch/cherrybot/tcp",
      "target": "https://api.interactions.ics.unisg.ch/cherrybot/tcp/target",
      "gripper": "https://api.interactions.ics.unisg.ch/cherrybot/gripper"
    },
  },
}


```

(b) *Endpoints* utilizados para la interacción con dispositivos de Internet de las Cosas

Figura 4.34: Información contenida en el archivo `EnglishConfig.json`

La clase `MainConfiguration` está equipada con un constructor que invoca al método `readConfigFile` para leer la información almacenada en el archivo de configuración principal. Una vez completada la configuración, se utiliza el método `closeConfigFile`. Además, se instancia la clase `Language` que, mediante el método `chooseLanguage`, permite al usuario seleccionar el idioma deseado.

Además, la clase `MainConfiguration` dispone de los métodos `loadWhisper` y `loadSpacy` para la configuración e inicialización de estos *frameworks*, respectivamente, según la elección del idioma. Este proceso se puede visualizar en las Figuras 4.35 y 4.36. Ambos métodos incorporan un bloque *try-catch* para gestionar posibles errores que puedan surgir durante la carga de cualquiera de los *frameworks*, informando al usuario en caso de alguna anomalía.



```

MainConfig.py 3 X
MainConfiguration > loadSpacy

def loadWhisper(self):

    #Initialize OpenVino module
    core = Core()
    #Get whisper data configuration
    whisper_data = self.config['WHISPER']

    try:

        #Load whisper model
        model = whisper.load_model(whisper_data['whisper_model'],
                                  device = whisper_data['device'])
        #Delete original whisper decoder and encoder modules
        del model.decoder
        del model.encoder

        #Charge new encoder and decoder modules
        model.encoder = wov.OpenVINOAudioEncoder(core,whisper_data['NEW_CONFIG']['encoder'])
        model.decoder = wov.OpenVINOTextDecoder(core, whisper_data['NEW_CONFIG']['decoder'])
        model.decode = partial(wov.decode, model)
        model.parameters = wov.parameters
        model.logits = partial(wov.logits, model)

    return model

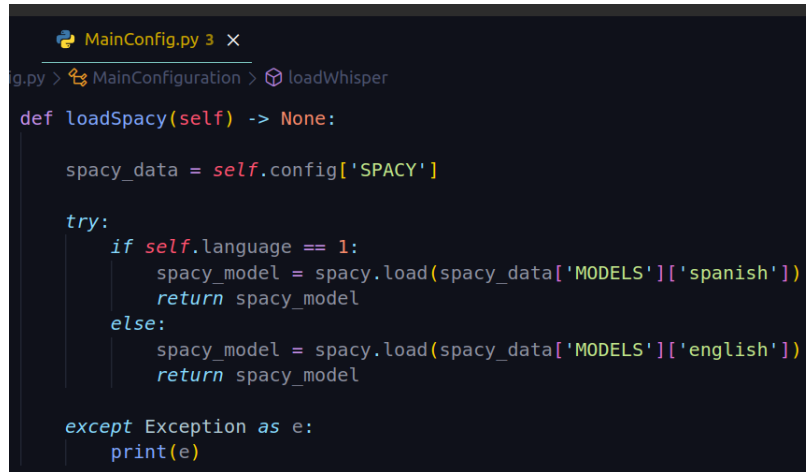
except Exception as e:
    print(e)

```

Figura 4.35: Método de la clase `MainConfiguration` para inicializar Whisper

En el proceso de inicialización de `Whisper`, se lleva a cabo un paso crucial que implica la sustitución de dos de sus clases fundamentales, `encoder` y `decoder`. Esta acción se realiza para integrar las nuevas clases, específicamente `OpenVINOAudioEncoder` y `OpenVINOTextDecoder`, que han sido adaptadas para incorporar `OpenVINO`. El objetivo de esta adaptación es optimizar el tiempo de traducción de archivos de audio.

Una vez que las nuevas clases mencionadas se han cargado con éxito, se considera que el proceso de inicialización de `Whisper` ha concluido. Este ajuste es esencial para aprovechar las mejoras proporcionadas por `OpenVINO` en la traducción de archivos de audio, contribuyendo significativamente a la eficiencia del sistema.



```

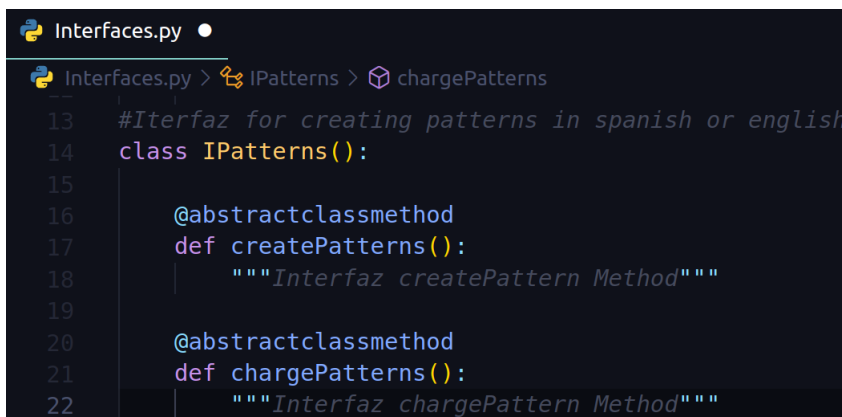
MainConfig.py 3 x
g.py > MainConfiguration > loadWhisper
def loadSpacy(self) -> None:
    spacy_data = self.config['SPACY']

    try:
        if self.language == 1:
            spacy_model = spacy.load(spacy_data['MODELS']['spanish'])
            return spacy_model
        else:
            spacy_model = spacy.load(spacy_data['MODELS']['english'])
            return spacy_model
    except Exception as e:
        print(e)

```

Figura 4.36: Método de la clase MainConfiguration para inicializar SpaCy

Como se observa en la Figura 4.36, para la inicialización de SpaCy solamente se requiere la identificación del idioma a utilizar para poder cargar el modelo adecuado que permita la obtención de la información requerida. Una vez que se ha concluido la inicialización de los *frameworks* requeridos, se realiza la implementación de las clases `EnglishConfig` y `SpanishConfig`, las cuales hacen uso de la interfaz `IPatterns` visualizada en la Figura 4.37. La interfaz incorpora los métodos `createPattern` y `chargePattern`. Estas funciones facilitan la creación de patrones para la identificación de información esencial. Dependiendo del idioma seleccionado, se elige la clase correspondiente con el propósito de cargar los patrones necesarios en el micro-servicio. Este proceso se explora con más detalle en la sección 4.3.7, donde se aborda la identificación de peticiones y la extracción de características.



```

Interfaces.py ●
Interfaces.py > IPatterns > chargePatterns
13 #Iterfaz for creating patterns in spanish or english
14 class IPatterns():
15
16     @abstractmethod
17     def createPatterns():
18         """Iterfaz createPattern Method"""
19
20     @abstractmethod
21     def chargePatterns():
22         """Iterfaz chargePattern Method"""

```

Figura 4.37: Interfaz utilizada para la generación de la clases `EnglishPattern` y `SpanishPattern`

Finalmente, todo este proceso es llevado a cabo mediante la utilización del método `loadModules` de la clase `MainConfiguration`, visualizado en la Figura 4.38. Este método hace un llamado a los métodos mencionados anteriormente para la inicialización de los *frameworks*, así como a los métodos `getModel` con el cual se carga el modelo utilizado para la identificación de usuarios y el método `getEndpoints`, utilizado para la obtención de los *endpoints* correspondientes para la comunicación con los sistemas externos, mediante el módulo de comunicación.

```

MainConfig.py 3
MainConfig.py > MainConfiguration > loadModules
89
90     def loadModules(self):
91         whisper_model = self.loadWhisper()
92         spacy_model = self.loadSpacy()
93         model = self.getModel()
94         endpoints = self.getEndpoints()
95
96         return whisper_model, spacy_model, model, endpoints

```

Figura 4.38: Método para inicialización de los módulos del micro-servicio

Para iniciar el micro-servicio utilizando Flask, es necesario ejecutar el comando `flask --app app.py run --host=0.0.0.0`. Después de ejecutar este comando, al usuario se le presentará un menú, tal como se muestra en la Figura 4.39, donde se le solicitará que elija el idioma para utilizar el micro-servicio. Una vez seleccionado el idioma deseado, el micro-servicio se inicializará y estará listo para su uso inmediato.

```

Selecciona idioma a utilizar
1.Español 2.Inglés
Choose language
1.Spanish 2.English
1
* Serving Flask app 'app'
* Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment.
* Running on all addresses (0.0.0.0)
* Running on http://127.0.0.1:5000
* Running on http://192.168.0.19:5000
Press CTRL+C to quit

```

Figura 4.39: Inicialización del micro-servicio en idioma español

4.3.4. Generación de repositorios

El micro-servicio gestiona tanto las solicitudes de los usuarios como las muestras necesarias para registrar a un nuevo usuario, utilizando un sistema de repositorios. Con el objetivo de facilitar la integración de nuevos tipos de repositorios en el futuro, se ha implementado el patrón de diseño *Factory*. Se creó la clase `RepositoryFactory` y la interfaz

IRepository para este propósito.

La clase `RepositoryFactory` cuenta con el método `buildRepository`, encargado de generar los repositorios correspondientes. Se han definido dos clases para la generación de repositorios: `Folder`, encargada de los repositorios para las peticiones de usuarios y cualquier otro repositorio necesario para el almacenamiento de información requerida por sistemas externos; y la clase `UserFolder`, diseñada para la generación de repositorios de usuarios registrados. En estos repositorios generados por la clase `UserFolder` se almacenan las muestras de audio solicitadas para la creación del modelo de identificación.

La implementación de este patrón de diseño se puede observar en la Figura 4.40. Esta estructura permite una fácil adaptación y expansión del sistema al incorporar nuevos tipos de repositorios, fortaleciendo la flexibilidad y escalabilidad del micro-servicio.

```

class UserFolder(IRepository):
    def __init__(self) -> None:
        self.audio_folder = Folder("audio")
        self.__AUDIO_FOLDER = "audio"
        self.__USER = 1
        self.__userpath = "user"+str(self.__USER)
        self.__USER_FOLDER = os.path.join(self.__AUDIO_FOLDER,self.__userpath)

    def createRepository(self)-> __name__:
        self.audio_folder.create_folder();

        while os.path.exists(self.__USER_FOLDER) == True:
            self.__USER +=1
            self.__userpath = "user"+str(self.__USER)
            self.__USER_FOLDER = os.path.join(self.__AUDIO_FOLDER,self.__userpath)

        os.mkdir(self.__USER_FOLDER)
        return self.__USER_FOLDER

#This is a design pattern that enables us to create different folders
class RepositoryFactory():
    @staticmethod
    def buildRepository(repository_type):
        if repository_type != "user":
            return Folder(repository_type).create_folder()
        elif repository_type == "user":
            return UserFolder().create_folder()

```

Figura 4.40: Implementación del patrón de diseño Factory para la generación de repositorios

Cuando se realiza la primera petición al micro-servicio, al almacenar la solicitud del usuario para su procesamiento, se crea un repositorio denominado *request*. En este repositorio se almacenan las peticiones y se continúan guardando las subsiguientes de los usuarios. Además, cuando se inicia el proceso de registro de un nuevo usuario, si es el primer registro, se genera un repositorio con el nombre `audio`. En este repositorio se crean subdirectorios correspondientes a los usuarios registrados. Estos subdirectorios se

nombran con el prefijo `user` seguido del número de usuario, como se ilustra en la Figura 4.41.

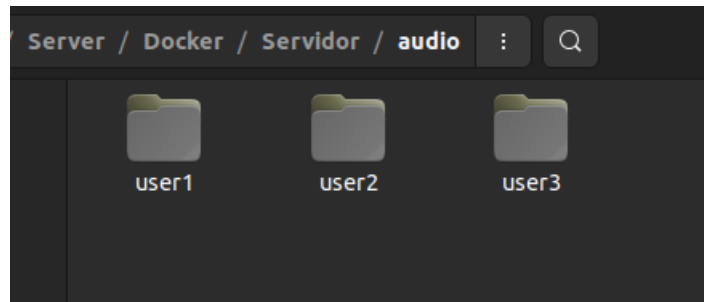


Figura 4.41: Generación de repositorios de usuarios para almacenar muestras de audio

Es posible generar nuevos repositorios para otros propósitos mediante la creación de una instancia de la clase `RepositoryFactory` y la utilización del método `createFolder`, al cual se le ingresa el nombre que se asignará al nuevo repositorio. Este enfoque brinda flexibilidad para la creación dinámica de repositorios según las necesidades específicas del sistema.

4.3.5. Manipulación y procesamiento de archivos de audio

Una vez que se recibe una petición desde cualquiera de las fuentes de entrada, como la aplicación Android o el sistema de captura permanente, es necesario generar un archivo de audio en formato `.wav` para facilitar su traducción a texto y la obtención de la información necesaria. El micro-servicio cuenta con la clase `AudioFile`, la cual se encarga de la generación, procesamiento y almacenamiento del archivo de audio en formato `.wav` mediante el método `saveAudio`, que a su vez contiene una instancia de la clase `FilterFactory` para el preprocesamiento del archivo de audio. El método `saveAudio` crea el archivo de audio en formato `.wav`, lo preprocesa y lo guarda en el repositorio `request`. Para la generación del archivo `.wav`, se utiliza la funcionalidad `export` proporcionada por `AudioSegment` de la biblioteca `pydub`.

```

def fileSegmentation(self, folder_path: __name__ ) -> None:
    __segmentation_time = 1 # Time in seconds that a file is going
    self.counter = 0

    filenames = os.listdir(folder_path)

    for file in filenames:
        self.counter += 1
        segment = AudioSegment.from_file(folder_path + "/" + file)
        batch_size = __segmentation_time * 1000
        duration = segment.duration_seconds
        batches = math.ceil(duration / __segmentation_time)

        inicio = 0
        j = 0;
        for i in range(batches):
            j = batches*(self.counter - 1) + i
            muestra = segment[inicio: inicio + batch_size]
            filename = f'/subsample_{j}.wav'
            muestra.export(folder_path+filename, format='wav')
            inicio+= batch_size

```

Figura 4.42: Método de la clase `AudioFile` para la segmentación de archivos de audio

En caso de una solicitud de registro, el micro-servicio recibe múltiples archivos de audio simultáneamente. Al igual que con el archivo de solicitud, es necesario convertirlos al formato `.wav`, preprocesarlos y luego segmentarlos en archivos de un segundo. Este proceso se lleva a cabo mediante los métodos `saveFiles` y `fileSegmentation`, el cual se presenta visualmente en la Figura 4.42. El método `saveFile` contiene una instancia de la clase `FilterFactory` para el preprocesamiento de los archivos de audio `.wav` antes de ser segmentados. Para la segmentación de los archivos recibidos se emplea la funcionalidad `fromfile` proporcionada por `AudioSegment` de la biblioteca `pydub`.

La clase `FilterFactory` brinda al micro-servicio la flexibilidad de la creación e implementación de múltiples filtros como filtros paso banda, paso bajo, etc. que permitan el preprocesamiento de los archivos de audio. Actualmente, la clase `FilterFactory` cuenta con la capacidad de implementar un filtro paso banda haciendo uso de la clase `BandFilter` mostrado en la Figura 4.43, que a su vez utiliza la interfaz `IFilter` para la implementación del método `createFilter`. Esta clase tiene como atributos privados, en su constructor, la declaración de las variables de la frecuencia de corte superior (`fc_high`), frecuencia de corte inferior (`fc_low`) y el orden del filtro (`order`).

```

class BandFilter(IFilter):
    def __init__(self) -> None:

        self.__fc_high = 7900
        self.__fc_low = 100
        self.__order = 5

    def create_filter(self, folder_path, audio_file, sample) -> None:

        audio_path = folder_path + audio_file

        sample_rate, audio_data = wavfile.read(audio_path)
        audio_data = audio_data.astype(np.float32) / np.max(np.abs(audio_data))
        duration = len(audio_data) / sample_rate
        time = np.linspace(0., duration, len(audio_data))
        nyquist = 0.5*sample_rate
        low = self.__fc_low/nyquist
        high = self.__fc_high/nyquist
        b, a = butter(self.__order, [low, high], btype='band')
        filtered_audio = filtfilt(b, a, audio_data)
        filtered_audio = filtered_audio / np.max(np.abs(filtered_audio))
        filtered_audio = (filtered_audio * 32767).astype(np.int16)
        wavfile.write(folder_path+audio_file, sample_rate, filtered_audio)

```

Figura 4.43: Clase implementada para la generación de filtro paso banda

El filtro paso banda es implementado una vez recibido el archivo de audio para la eliminación de las frecuencias altas y bajas que generan ruido en el archivo. De esta manera, se mejora la calidad del audio para obtener mejores resultados en la identificación del usuario y en la traducción de audio a texto.

Para determinar la frecuencia de corte superior, se consideró la frecuencia de *Nyquist*, que representa la frecuencia máxima recuperable de una señal de audio dada una tasa de muestreo fija [Pérez, 2011]. En términos generales, puede definirse como la mitad de la frecuencia de muestreo utilizada para grabar una señal, aunque se encuentra debidamente especificada en la Sección 2.1.1 del Capítulo 2. Con base en la frecuencia de *Nyquist* y la tasa de muestreo utilizada en las fuentes de entrada para la captura de audio (igual a 16 kHz), se determinó que la frecuencia de corte superior máxima a utilizar es de 8 kHz.

Por otro lado, la frecuencia de corte inferior se estableció considerando el rango de frecuencias más bajo que el oído humano es capaz de percibir, el cual oscila entre 20 Hz y 20 kHz [González, 2018]. Para definir el orden del filtro, se llevaron a cabo pruebas aplicando órdenes del 1 al 10. Sin embargo, se determinó que la implementación de filtros de orden superior a 6 resultaba en la eliminación de gran parte de la información contenida en el archivo de audio, dificultando la traducción a texto. No obstante, se observó que el filtro de orden cinco permitía mejorar la calidad del audio, obteniendo resultados más

satisfactorios en la traducción a formato de texto. Para la generación del filtro paso banda se utilizó la biblioteca `SciPy`, la cual cuenta con diferentes funcionalidades dedicadas al procesamiento de señales.

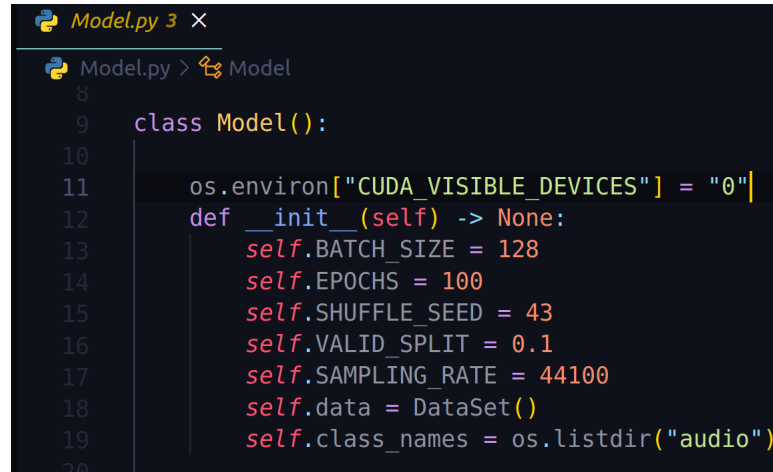
4.3.6. Generación del modelo de identificación de usuarios

Para la generación del modelo de identificación de usuarios, se hizo uso del proyecto *Speaker Recognition*, presentado por `Keras` para la identificación de usuarios, mediante el uso de la voz. Se tomó la decisión de utilizarlo, para su evaluación, en la implementación del módulo de identificación de usuarios del micro-servicio, ya que se ha reportado en su página oficial², que el modelo puede llegar a obtener una exactitud del 98 % para la identificación de cinco personas.

Este modelo utiliza una red convolucional de una dimensión (Red Convolucional 1D), ya que es un tipo de arquitectura de red neuronal diseñada para trabajar con datos unidimensionales como: señales de audio, señales biológicas, etc. La red implementada para la generación del modelo está basada en bloques residuales o también conocida como `ResNet`. A grandes rasgos, es un tipo de red que contiene conexiones denominadas conexiones residuales o *skip connections*, las cuales ayudan a saltar entre una o más capas, permitiendo que el modelo aprenda una representación residual, en lugar de forzar a cada capa a aprender la representación completa desde cero. Esta red convolucional recibe como datos de entrada la Transformada Rápida de *Fourier* (FFT, por sus siglas en inglés) de cada muestra de audio perteneciente al conjunto de datos de entrenamiento

Para la implementación de este modelo de identificación de personas, se realizaron adecuaciones en la distribución del proyecto mencionado. Se elaboraron dos clases, la primera denominada `Model` encargada de la generación del modelo y su implementación, y la segunda denominada `DataSet` responsable de la generación del conjunto de datos de entrenamiento y del conjunto de datos de validación, así como de la generación de los datos de entrada de la red neuronal convolucional, obteniendo la Transformada Rápida de *Fourier* de cada uno de los datos de entrenamiento.

²https://keras.io/examples/audio/speaker_recognition_using_cnn/

The image shows a code editor window titled 'Model.py 3'. The code defines a class 'Model' with an '__init__' method. Inside the method, it sets the environment variable 'CUDA_VISIBLE_DEVICES' to '0'. It then initializes several class attributes: 'BATCH_SIZE' (128), 'EPOCHS' (100), 'SHUFFLE_SEED' (43), 'VALID_SPLIT' (0.1), and 'SAMPLING_RATE' (44100). Finally, it creates a 'DataSet' object and lists the contents of the 'audio' directory as 'class_names'.

```
8
9 class Model():
10
11     os.environ["CUDA_VISIBLE_DEVICES"] = "0"
12     def __init__(self) -> None:
13         self.BATCH_SIZE = 128
14         self.EPOCHS = 100
15         self.SHUFFLE_SEED = 43
16         self.VALID_SPLIT = 0.1
17         self.SAMPLING_RATE = 44100
18         self.data = DataSet()
19         self.class_names = os.listdir("audio")
20
```

Figura 4.44: Constructor de la clase `Model` para la generación del modelo de identificación de usuarios

Como se puede observar en la Figura 4.44, la clase `Model` contiene un constructor en donde se genera una instancia de la clase `DataSet` para el procesamiento del conjunto de datos, así mismo se definen parámetros como:

1. *Batch_size*: número de muestras de datos que se utilizan en una iteración para actualizar los pesos.
2. *Epochs*: número de épocas de entrenamiento de la red.
3. *Shuffle_Seed*: semilla para el generador de números aleatorios utilizado para mezclar el conjunto de datos, antes de cada época, durante el entrenamiento.
4. *Valid_Split*: proporción del conjunto de datos que se debe reservar para la validación durante el entrenamiento, siendo 1 el valor correspondiente al 100%.
5. *Samplig_Rate*: especifica la frecuencia de muestreo utilizada al adquirir los datos de audio.

Una vez que se han segmentado las muestras de audio del usuario a registrar, se utiliza el método `trainModel` de la clase `Model`, para comenzar el entrenamiento del modelo de identificación de usuarios. Este método aprovecha todas las funcionalidades de la clase `DataBase`. Para la generación del conjunto de datos de entrenamiento y el de validación se utiliza el método `classifyDataset`. Este método organiza las rutas de archivo y las etiquetas en función de las clases presentes en el directorio especificado por `audio_path`. En este contexto, las clases representan a los usuarios que tienen un repositorio con sus muestras de audio (e.g., la clase 1 es igual al repositorio `user1`). Cada archivo de audio se asigna a una clase específica y se almacena junto con su etiqueta correspondiente en las

listas `audio_paths` y `labels`.

Posteriormente, se realiza una mezcla de todos los archivos de audio, junto con sus etiquetas asociadas al conjunto de datos, para introducir aleatoriedad en el orden de los datos durante el entrenamiento de la red. Una vez que se han mezclado todos los archivos de audio con sus respectivas etiquetas, se generan los conjuntos de datos de entrenamiento y de validación tomando en consideración el atributo `Valid.Split` de la clase `Model`, el cual designa el porcentaje indicado al conjunto de datos de validación y el restante al conjunto de datos de entrenamiento (e.g, Si `Valid.Split` es igual a 0.1, se asigna el 10 % del conjunto de datos total al conjunto de validación y el 90 % al conjunto de datos de entrenamiento).

Con ambos conjuntos de datos generados, se aplica el método `AudioToFFT` de la clase `DataSet` para calcular la Transformada Rápida de *Fourier* de cada archivo. Esta transformada sirve como entrada para el entrenamiento del modelo y su posterior validación. Posteriormente se procede a la generación de la arquitectura de la red convolucional, llamando al método `buildModel` al cual se le ingresa, como parámetros de entrada, la variable `SamplingRate` y el número de clases (usuarios), y haciendo uso del método `residual_block`.

Como se puede observar en la Figura 4.45, el método `residual_block` define la estructura interna de cada bloque, donde se especifica el número de filtros a emplear en las convoluciones. Cada bloque consta de una conexión de atajo ("*shortcut*") creada mediante una convolución 1D con un solo filtro de tamaño uno, preservando así la información original. Dentro del bloque, se aplica un bucle de convoluciones 1D seguidas de activaciones `ReLU`, con el objetivo de aprender representaciones más complejas. El bloque se concluye con una convolución final y la suma de la salida con el *shortcut*, seguida de otra activación `ReLU`. Además, se incorpora una capa de `MaxPooling 1D` para reducir las dimensiones espaciales y mejorar la eficiencia computacional.


```

def residual_block(self,x, filters, conv_num=3, activation="relu"):
    # Shortcut
    s = keras.layers.Conv1D(filters, 1, padding="same")(x)
    for i in range(conv_num - 1):
        x = keras.layers.Conv1D(filters, 3, padding="same")(x)
        x = keras.layers.Activation(activation)(x)
    x = keras.layers.Conv1D(filters, 3, padding="same")(x)
    x = keras.layers.Add()([x, s])
    x = keras.layers.Activation(activation)(x)
    return keras.layers.MaxPool1D(pool_size=2, strides=2)(x)

```

Figura 4.45: Arquitectura del bloque residual utilizado en la arquitectura RestNet de la red convolucional

En el método `build_model`, visualizado en la Fig 4.46, se utilizan estos bloques residuales de manera secuencial, variando el número de filtros y repeticiones para construir una red más profunda. Por ejemplo, se comienza con un bloque de 16 filtros y 2 repeticiones, luego se utiliza un bloque de 32 filtros y 2 repeticiones, y así sucesivamente. La salida de cada bloque residual se convierte en la entrada del siguiente bloque. Después de los bloques residuales, se incorpora una capa de `AveragePooling 1D` con un tamaño de `pool` y un `stride` de 3 para reducir aún más las dimensiones espaciales de la salida. La red continúa con capas totalmente conectadas, utilizando activación `ReLU`, con 256 y 128 unidades respectivamente. Finalmente, la capa de salida consiste en una capa `Dense` con activación `softmax`, indicando la probabilidad de pertenencia a cada clase (usuario).

Finalmente, ya que se ha definido la arquitectura de la red convolucional, se inicia el entrenamiento del modelo mediante la función `fit`, la cual toma como parámetros de entrada el conjunto de datos de entrenamiento y de validación, número de épocas y, si se desea, parámetros adicionales como algunas métricas para decidir cuando detener el entrenamiento, sin la necesidad de aplicar todas las épocas. En este caso se utiliza la variable `earlystopping` que le indica al sistema que si después de 20 épocas no hay mejora en el modelo se termine el entrenamiento. Para utilizar el modelo, se hace uso del método `use_model`, el cual tiene como parámetro de entrada el archivo de audio a evaluar para la identificación del usuario que ha realizado la petición. Se ha definido un umbral (*Threshold*) de 0.8, esto quiere decir que si se obtienen valores por debajo, el micro-servicio no ha identificado al usuario.

```

def build_model(self, input_shape, num_classes):
    inputs = keras.layers.Input(shape=input_shape, name="input")

    x = self.residual_block(inputs, 16, 2)
    x = self.residual_block(x, 32, 2)
    x = self.residual_block(x, 64, 3)
    x = self.residual_block(x, 128, 3)
    x = self.residual_block(x, 128, 3)

    x = keras.layers.AveragePooling1D(pool_size=3, strides=3)(x)
    x = keras.layers.Flatten()(x)
    x = keras.layers.Dense(256, activation="relu")(x)
    x = keras.layers.Dense(128, activation="relu")(x)

    outputs = keras.layers.Dense(num_classes, activation="softmax",
    return keras.models.Model(inputs=inputs, outputs=outputs)

```

Figura 4.46: Arquitectura ResNet de la red convolucional para la generación del modelo de identificación de personas

Se llevaron a cabo las primeras pruebas para la generación del modelo de identificación de usuarios, en las cuales se midió el tiempo que el modelo tarda en completar su entrenamiento. Estas pruebas se ejecutaron en una computadora equipada con un procesador Intel Core i5 de novena generación, con cuatro núcleos, 8 GB de memoria RAM y una GPU GEFORCE GTX 1250. Para la realización del entrenamiento del modelo, se empleó la GPU con el objetivo de comparar los tiempos de entrenamiento.

Las pruebas consistieron en entrenar modelos para identificar desde dos hasta seis personas, generando 100 muestras de audio de un segundo por cada individuo. La exactitud del modelo se evaluó utilizando un conjunto de datos de validación generado por la clase `Model`. Los resultados obtenidos se presentan en la Tabla 4.8.

Tabla 4.8: Resultados de las pruebas de velocidad y exactitud del entrenamiento del modelo de identificación de personas

Número de usuarios	Tiempo de entrenamiento [min]	Exactitud
2	5.06	1
3	15.90	1
4	16.41	1
5	17.30	1
6	23.95	0.98

Es evidente que el entrenamiento del modelo es computacionalmente costoso, incluso al utilizar una GPU. Por ejemplo, el entrenamiento de un modelo para identificar a dos personas tomó cinco minutos, mientras que para un modelo que pueda identificar seis personas, el entrenamiento llevó más de 20 minutos. Esto indica que realizar el entrenamiento de un nuevo modelo para el registro de un usuario en el micro-servicio no es factible. Dado que el dispositivo estaría constantemente en uso atendiendo peticiones, no sería capaz de llevar a cabo ambas acciones simultáneamente. Por lo tanto, se tomó la decisión de utilizar un dispositivo externo al micro-servicio, que permita realizar el entrenamiento sin la necesidad de involucrar al micro-servicio y, una vez que se haya generado el nuevo modelo, cargarlo en el micro-servicio para su uso.

Se procedió a evaluar el rendimiento del modelo desarrollado para la identificación de dos personas mediante la construcción de una matriz de confusión. Los resultados de estas evaluaciones se presentan en la Tabla 4.9. En total, se llevaron a cabo 40 pruebas para la identificación de los usuarios.

Tabla 4.9: Matriz de confusión para la identificación de dos usuarios

	Usuario 1	Usuario 2
Usuario 1	18	2
Usuario 2	12	8

$$\text{Precisión} = \frac{18}{18 + 2} = 0,9 \quad (4.5)$$

$$\text{Especificidad} = \frac{8}{8 + 12} = 0,4 \quad (4.6)$$

$$\text{Exactitud} = \frac{18 + 8}{18 + 2 + 12 + 8} = 0,65 \quad (4.7)$$

La evaluación de los resultados, tal como se refleja en las ecuaciones 4.5, 4.6, y 4.7, revela que: 1) los resultados de la exactitud obtenida con el conjunto de datos de validación, obtenidos al entrenar el modelo de identificación de usuarios, es incorrecto ya que el modelo desarrollado mostró limitaciones en su capacidad de identificación, y 2) aunque la precisión sugiere que el sistema logra reconocer, de manera acertada, al usuario número uno durante sus solicitudes, la medida de exactitud revela deficiencias significativas. El valor de exactitud indica que el modelo, en su mayoría, identifica al usuario número dos como si fuera el usuario número uno y, rara vez, logra una identificación precisa del usuario número dos. Este desequilibrio se traduce en un valor de sensibilidad del sistema inferior al 50 %, indicando una baja capacidad para identificar al usuario número dos de

manera efectiva.

Los factores que se detectaron como causantes de este primer resultado fueron: 1) la cantidad de muestras de audio no son suficientes para poder generar un buen modelo de identificación de usuarios, 2) una gran cantidad de las muestras de audio de un segundo no son de gran utilidad para en entrenamiento del modelo, ya que no contenían muestras de la voz del usuario, sino sólomente contenían ruido, indicando que el usuario se mantenía callado por periodos, afectando el entrenamiento del modelo y 3) las voces de los usuarios son similares, lo cual causa que los valores de la Transformada Rápida de *Fourier* obtenida de cada muestra de audio sean similares, complicando la generación del modelo y, por ende, la correcta identificación de los usuarios.

Se llevaron a cabo pruebas adicionales, ampliando el conjunto de datos para el entrenamiento. Se solicitó a los usuarios la generación de cinco muestras de audio, cada una con una duración de tres minutos, con el objetivo de crear un conjunto de datos compuesto por 450 muestras por cada usuario. Además, se pidió a los usuarios que mantuvieran una conversación constante para minimizar la probabilidad de que, durante la segmentación de las muestras en archivos de audio de un segundo, se generaran archivos que carecieran de fragmentos de la voz del usuario. Las pruebas fueron realizadas a tres personas que no tuvieran una gran similitud entre sus voces. Los resultados obtenidos se muestran en la Tabla 4.10.

Tabla 4.10: Matriz de confusión para la identificación de tres usuarios

	Usuario 1	Usuario 2	Usuario 3
Usuario 1	19	1	0
Usuario 2	3	16	1
Usuario 3	0	2	18

- Resultados Usuario 1:

$$\text{Precisión Usuario 1} = \frac{19}{19 + 1 + 0} = \frac{19}{20} = 0,95 \quad (4.8)$$

$$\text{Especificidad Usuario 1} = \frac{34}{34 + 1 + 2} = \frac{34}{37} = 0,92 \quad (4.9)$$

- Resultados Usuario 2:

$$\text{Precisión Usuario 2} = \frac{16}{3 + 16 + 1} = \frac{16}{20} = 0,8 \quad (4.10)$$

$$\text{Especificidad Usuario 2} = \frac{33}{33 + 1 + 3} = \frac{33}{37} = 0,89 \quad (4.11)$$

- Resultados Usuario 3:

$$\text{Precisión Usuario 3} = \frac{18}{0 + 2 + 18} = \frac{18}{20} = 0,9 \quad (4.12)$$

$$\text{Especificidad Usuario 3} = \frac{34}{34 + 2 + 3} = \frac{34}{39} = 0,87 \quad (4.13)$$

- Exactitud del modelo de reconocimiento de usuarios:

$$\text{Exactitud} = \frac{19 + 16 + 18}{19 + 1 + 3 + 16 + 2 + 18 + 1} = \frac{53}{60} \approx 0,88 \quad (4.14)$$

Como se puede observar en la Tabla 4.10, al incrementar la cantidad de muestras por usuario para el entrenamiento del modelo encargado de la identificación de los usuarios y mejorando la calidad de las muestras, al reducir los momentos en que el usuario se mantiene en silencio, el módulo de identificación de usuarios es capaz de realizar mejores predicciones de quién es el usuario que realiza la petición. Como se observa en las métricas mostradas de la Ecuación 4.8 a la Ecuación 4.14, los valores incrementaron, obteniendo valores en la especificidad superiores a lo obtenido en la Ecuación 4.6 referente a las primeras evaluaciones realizadas. Sin embargo se debe tener en consideración que aún existen algunos problemas al momento de la identificación de usuarios con voces similares, ya que el sistema puede llegar a confundirse, como se observa en los valores de la Tabla 4.10 en donde al usuario 2 se le reconoció como usuario 1 en tres ocasiones y al usuario 3 como el usuario 2 en dos ocasiones.

4.3.7. Identificación de petición y extracción de información relevante

Como se mencionó en la Sección 4.3.3, una vez que se han inicializado los *frameworks* requeridos, se implementan las clases `EnglishConfig` y `SpanishConfig`, las cuales utilizan la interfaz `IPatterns`. Estos archivos contienen una serie de patrones generados bajo el concepto de coincidencias basadas en reglas y se encuentran descritos en el método `createPatterns`. Permiten la extracción de la información relevante que se desea obtener, dependiendo de los requerimientos del sistema externo. Estos patrones ayudan a la identificación tanto de palabras como de oraciones, haciendo uso de un diccionario con las palabras u oraciones que se desean identificar. Para la generación de estos patrones se requiere de la especificación de parámetros que ayuden al *framework* a realizar una identificación mas rápida de los elementos solicitados.

Dentro de los parámetros que se pueden utilizar se encuentran: 1) manera en que se analiza la palabra (e.g., “LOWER”, indica que la palabra deberá buscarse en minúsculas,

por lo cual las palabras analizadas serán transformadas a minúsculas para su análisis), 2) palabra a identificar, 3) especificación de cuantas veces se desea identificar la palabra (e.g., OP:“?” indica que la palabra debe identificarse sólo una vez en la oración), 4) tipo de palabra (e.g., “POS”：“VERB”, indica que la palabra a buscar deberá ser un verbo dentro del texto analizado) etc. Si se desea agregar más de una palabra al patrón, se debe hacer el mismo procedimiento.

La Figura 4.47 muestra un ejemplo de la generación de un patrón cuyo objetivo es la identificación de las palabras *at* y *to* una sola vez dentro de una oración, seguido de la identificación de uno o más números.

```
numbers_pattern = [{'LOWER':{'IN':['at','to']},'OP':'?'},{ 'POS':'NUM','OP':'+'}]
```

Figura 4.47: Ejemplo de la generación de un patrón para la identificación de elementos, utilizando el *framework* SpaCy

Una vez que los patrones necesarios para la extracción de información han sido definidos, generados y cargados, mediante los métodos `createPatterns` y `chargePatterns`, respectivamente, se emplea el método `audioToText` de la clase `RelevantInformation` para la traducción del archivo de audio a texto. Después, se utiliza el método `getIntention` para la identificación del dispositivo con el cual interactuar. Finalmente, mediante la clase `DeviceFactory`, se genera una instancia de la clase referente a dicho dispositivo. Esto implica la necesidad de crear una clase específica para cada sistema externo, permitiendo así la obtención de la información que será enviada por el módulo de comunicación.

Dichas clases utilizan la interfaz `IDevice`, que contiene el método `getFeatures` que, a su vez, hace uso del método `createJson`, encargado de la generación del formato *json* requerido. El formato *json* correspondiente para la interacción con cada dispositivo externo se toma del archivo de configuración `EnglishConfig.json` o `SpaishConfig.json` dependiendo del idioma, los cuales han sido cargados en la inicialización del micro-servicio.

Este enfoque proporciona una estructura modular y eficiente para la extracción de información específica de cada sistema externo, a través del micro-servicio. La Figura 4.48 muestra la implementación de la clase `DeviceFactory`.

```
class DeviceFactory():
    @staticmethod
    def createDevice(device):
        if device == "robot":
            return Robot().getFeatures()
        elif device == "lamp":
            return Lamp().getFeatures()
        elif device == "blinds":
            return Blind().getFeatures()
```

Figura 4.48: Implementación del patrón de diseño Factory para la generación de instancias de dispositivos

4.3.8. Envío de comandos

Una vez que se ha generado el formato *json* en el módulo de procesamiento de lenguaje natural, este es transmitido al módulo de comunicación a través de una instancia de la clase `CommunicationFactory`. Esta clase sigue el patrón de diseño *Factory* y su implementación tiene como objetivo proporcionar una mayor flexibilidad al micro-servicio. Esto le permite abordar diversos requisitos de comunicación para la interacción con sistemas externos, de manera eficiente.

Actualmente, el micro-servicio es capaz de implementar dos métodos de comunicación. El primero se realiza mediante el protocolo de comunicación HTTP, basado en el modelo de solicitud y respuesta, a través de la clase `IoTCommunication`. El segundo método se fundamenta en el concepto de productor-consumidor, utilizando colas de mensajes o eventos. Para ello, se hace uso del *framework* `Kafka`, gestionado por la clase `BrokerCommunication`. Esta diversidad en las opciones de comunicación amplía la capacidad del micro-servicio para interactuar, de manera efectiva, con diversos sistemas externos.

La clase `IoTCommunication` utiliza la librería `request` para la implementación de métodos basados en la arquitectura RESTful (POST, GET, PUT y DELETE). Estos métodos facilitan el envío de información a dispositivos externos, aprovechando los *endpoints* predefinidos durante la inicialización del micro-servicio, mediante los archivos `EnglishConfig.json` o `SpanishConfig.json`, dependiendo del idioma seleccionado.

```

class BrokerCommunication():
    def Producer(self, message):

        message_encoded = message.encode('utf-8')
        bootstrap_servers = ['localhost:29092']
        topicName = 'core'
        producer = KafkaProducer(bootstrap_servers=bootstrap_servers)
        try:
            ack = producer.send(topicName, message_encoded)
            metadata = ack.get()
        except Exception as e:
            print(e)
    def Consumer(self):

        topic = 'core'
        bootstrap_servers = 'localhost:29092'
        partition = 0
        consumer = KafkaConsumer(bootstrap_servers=bootstrap_servers)
        topic_partition = TopicPartition(topic, partition)
        consumer.assign([topic_partition])
        consumer.seek_to_end(topic_partition)

        for message in consumer:
            return message.value.decode('utf-8')

```

Figura 4.49: Implementación de la clase BrokerCommunication

Por otro lado, la clase `BrokerCommunication` visualizada en la Figura 4.49 utiliza la biblioteca `kafka` y presenta dos métodos para facilitar la comunicación con sistemas externos. Estos métodos son: `producer`, que se encarga del envío de información y `consumer`, diseñado para la lectura de respuestas provenientes de un sistema externo.

```

(ser) ivan@valdespin:~/Server/Docker/Serveridor$ flask --app app run
2023-06-11 20:36:28.815738: I tensorflow/core/platform/cpu_feature_gua
To enable the following instructions: AVX2 FMA, in other operations, r
2023-06-11 20:36:29.750322: W tensorflow/compiler/tf2tensorrt/utils/py
* Serving Flask app 'app'
* Debug mode: off
WARNING: This is a development server. Do not use it in a production d
* Running on all addresses (0.0.0.0)
* Running on http://127.0.0.1:5000
* Running on http://192.168.0.16:5000
Press CTRL+C to quit
2023-06-11 20:37:40.668914: E tensorflow/compiler/xla/stream_executor/
2023-06-11 20:37:40.668946: I tensorflow/compiler/xla/stream_executor/
2023-06-11 20:37:40.668959: I tensorflow/compiler/xla/stream_executor/
2023-06-11 20:37:40.668988: I tensorflow/compiler/xla/stream_executor/
2023-06-11 20:37:40.669032: I tensorflow/compiler/xla/stream_executor/
2023-06-11 20:37:40.669056: I tensorflow/compiler/xla/stream_executor/
1/1 [=====] - 0s 496ms/step
[0.01872891 0.05178086 0.9294902 ]
Predicted: user3

(ser) ivan@valdespin:~/kafka/bin$ ./kafka-console-consumer.sh --bootstrap-ser
ver localhost:9092 --topic server2 --partition 0
Esto es una prueba de conexión.

```

Figura 4.50: Prueba de comunicación del micro-servicio con un sistema *broker* utilizando el framework Kafka

La Figura 4.50 muestra una de las pruebas realizadas para evaluar la comunicación, utilizando la clase `BrokerCommunication` entre el micro-servicio y un *broker* implementado localmente. Como se puede observar en la prueba, el usuario ha enviado una petición

mediante la aplicación Android, la cual en este momento sólo es la oración “Esto es una prueba de conexión”. El micro-servicio logra identificar al usuario como usuario 3 y envía la petición al sistema *broker* mediante la clase **BrokerCommunication**. Como se observa, el sistema *broker* logra recibir de manera adecuada la petición del usuario, con lo cual se corrobora la correcta comunicación entre ambos sistemas.

Capítulo 5

Casos de estudio

En este Capítulo se describen los casos de estudio de la presente propuesta. La Sección 5.1 expone el primer caso de estudio sobre un compañero digital para el cuidado de la diabetes. En la Sección 5.2, se describe otro caso de estudio referente a un sistema para la interacción con dispositivos del Internet de las Cosas. Finalmente, en la Sección 5.3, se explica el último caso de estudio que trata de la integración del sistema de reconocimiento de voz y habla a un sistema de identificación de objetos basado en contexto.

5.1. Compañero digital para el cuidado de la diabetes

En este apartado, se aborda el primer caso de estudio para el sistema de reconocimiento de voz y habla multi-fuente. En la Sección 5.1.1 se realiza una descripción general del caso de estudio seguido de la Sección 5.1.2 en donde se describe la adaptación realizada al sistema de reconocimiento de voz y habla multi-fuente para abordar este caso de estudio.

5.1.1. Descripción general

La diabetes es una enfermedad crónica que afecta a un gran porcentaje de la población mundial [Roca et al., 2021], [INEGI, 2021]. En México, la diabetes mellitus es la tercera causa de defunciones, superada sólo por el COVID-19 y enfermedades cardíacas, afectando en su mayoría a personas mayores [INEGI, 2021]. Al ser una enfermedad crónica, que se divide en varias etapas, se requiere de múltiples cuidados, dependiendo de la etapa, que ayuden a la persona a poder llevar una vida lo más sana posible. Entre los cuidados generales que se requieren para cualquier etapa, se encuentran: adoptar una alimentación y un peso saludable, realizar actividades físicas, monitorear la glucosa en la sangre y llevar un control sobre el fármaco recetado por el médico [OPS, 2021].

El primer caso de estudio se inscribe en la propuesta doctoral de desarrollo de una arquitectura de compañero digital para pacientes con alguna enfermedad crónica. El compañero digital está basado en la integración de servicios en línea y la interconexión de infraestructura de cómputo ubicuo, con el fin de explotar los datos generados en beneficio de los usuarios.

El compañero digital tiene como característica diferencial el cambio de paradigma de diseño, pasando del enfoque tradicional de sistemas basados en Interacción Humano-Computadora hacia el paradigma de Cómputo Ubicuo. Esto es que de un sistema que es casi totalmente reactivo a las órdenes de los usuarios, se tiene un sistema que centra su atención en el usuario y es proactivo en mayor medida, haciéndose pasar desapercibido y ayudando a los pacientes en la gestión de tareas de cuidados de la salud que saturan el día a día de este tipo de usuarios.

Como se puede observar en la Figura 5.1, la arquitectura del compañero digital está diseñada bajo el principio de alta cohesión y bajo acoplamiento [Larman, 2005] y está constituida por ocho componentes esenciales, en donde cada uno de ellos es un servicio web. Además, se planea que sean concebidos como micro-servicios para poder obtener las propiedades de tolerancia a fallas y extensibilidad. Los micro-servicios que conforman la arquitectura son: 1) *Broker Service*, 2) *Speech Recognition Service*, 3) *Behavioral Intelligence Service*, 4) *Biometric Monitoring Service*, 5) *Semantic Web Service*, 6) *Proactive Response Service*, 7) *Knowledge Base* y 8) *Digital Companion Service*.

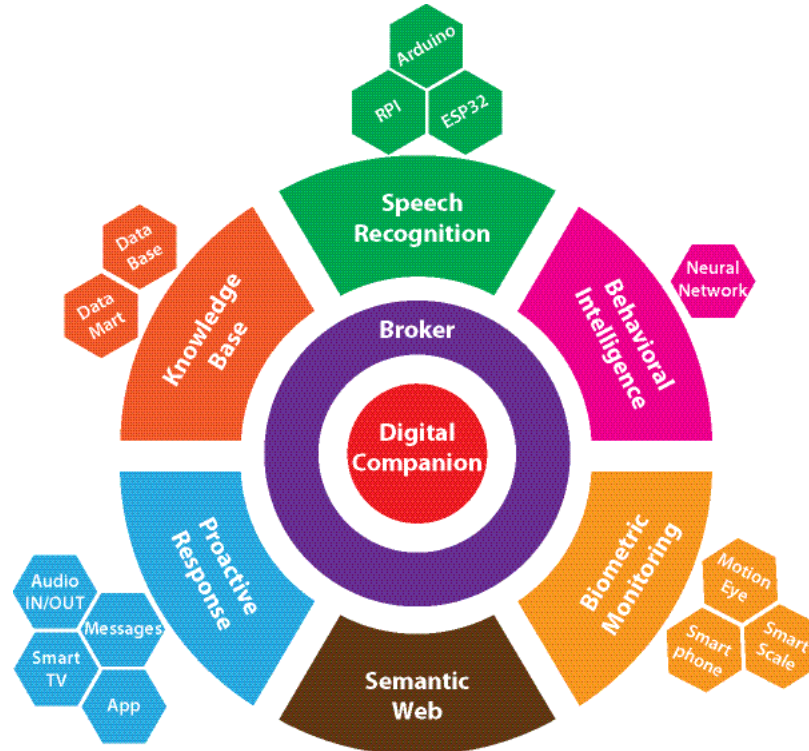


Figura 5.1: Diagrama general de la arquitectura de compañeros digitales para la atención a pacientes con enfermedades crónicas.

Cada uno de los micro-servicios tiene responsabilidades específicas definidas, las cuales son:

1. **Broker Service:** es el encargado de enrutar los eventos a través de un sistema de colas de peticiones que gestionan los mensajes con una estructura definida para facilitar el intercambio de información entre diversos micro-servicios.
2. **Speech Recognition Service:** es el responsable de recibir las señales de audio e identificar algunas frases clave (comandos) para realizar acciones solicitadas por parte del usuario. Tiene como principales responsabilidades: reconocer comandos del usuario y enviar las peticiones identificadas al servicio correspondiente, mediante el envío de un evento a través del *Broker Service*.
3. **Behavioral Intelligence Service:** tiene como función la de recibir datos del *Biometric Monitoring Service* para su análisis con métodos de *Machine Learning*. Este servicio tiene la posibilidad de acelerar el entrenamiento de los modelos de *Machine Learning*.
4. **Biometric Monitoring:** este servicio cuenta con un *middleware* que integra los diversos dispositivos en la infraestructura disponible para el entorno del usuario.

Se recolectan y estructuran los patrones biométricos y de comportamiento que sean captados; los datos recolectados son enviados al servicio *Behavioral Intelligence* para su análisis y generación de conocimiento.

5. ***Semantic Web Service:*** este servicio es capaz de recibir consultas y responderlas, a partir de la base de conocimiento del compañero digital. En caso de no encontrar alguna respuesta en la base de conocimiento, se realizará una búsqueda en la Web para responder y retroalimentar a la base de conocimiento.
6. ***Proactive Respose Service:*** este servicio es el encargado de realizar la interacción con el usuario para dar respuesta a sus peticiones y sugerencias o información, de manera proactiva. Así mismo, este servicio apoya al usuario en el cambio progresivo de hábitos de comportamiento que le permitan avanzar en el control de su estado de salud.
7. ***Digital Companion Service:*** este servicio es el responsable de reunir los datos, información y conocimiento para generar un modelo de retroalimentación, con el objetivo de adaptarse al usuario. También es el encargado de establecer comunicación con los sistemas gestores de citas, de consolidar el seguimiento médico y de establecer estrategias de adherencia a la medicación del usuario acorde con su perfil.

Los casos de uso que tiene este compañero digital son:

1. Gestión de citas médicas.
 - Análisis clínicos.
 - Revisión médica programada.
 - Revisión de emergencia.
2. Seguimiento médico.
 - Envío de parámetros al médico.
 - Análisis y visualización de parámetros.
 - Retroalimentación de resultados o sugerencias.
 - Recomendaciones por el compañero digital.
3. Apoyo al paciente para la adherencia de toma de medicamentos
 - Ingreso de prescripción médica.
 - Acceso a un calendario que recuerde la toma de medicamentos e informe sobre la forma correcta de la ingesta o administración del medicamento.
 - Recomendación educacional sobre los efectos del medicamento o contraindicaciones.

- Registro de emociones para reportar al médico y búsqueda de conocimiento mediante *Machine Learning*.

El sistema de reconocimiento de voz y habla, diseñado en el Capítulo 3, cumple con las funciones del micro-servicio de *Speech Recognition* que se muestra en la Figura 5.1, enfocándose en personas que padecen de diabetes, ya que es uno de los principales problemas médicos en México. Este micro-servicio es capaz obtener la información requerida por el micro-servicio *Broker Service* para la atención de peticiones de usuarios, en el idioma español, respecto a los siguientes tópicos:

- Gestión de citas con el médico.
- Recordatorio de toma de medicamento.
- Búsqueda de información referente a la diabetes.

5.1.2. Identificación de peticiones y extracción de información para la comunicación con el *Broker Service*

La implementación realizada para abordar este caso de estudio consta de tres pasos, los cuales son: 1) detección del tipo de petición del usuario, 2) obtención de información relevante y 3) generación y envío del formato *json* para la comunicación con el *Broker Service*.

Para la detección del tipo de petición solicitada por el usuario, se hace uso de una función de **SpaCy** denominada Similaridad Semántica o *Semantic Similarity* que, en el idioma español, puede ser utilizada solamente por los modelos `es_core_news_md` y `es_core_news_lg`. Esta funcionalidad permite medir la similitud entre dos textos en términos de su significado. Utiliza modelos de lenguaje preentrenados para representar las palabras y frases en un espacio vectorial semántico, donde las similitudes se calculan mediante medidas como la similitud de coseno [Honnibal et al., 2020a]. Mientras más cercanos estén los vectores de representación de dos textos, mayor será su similitud semántica.

Se generaron tres oraciones correspondientes a cada tipo de petición, contenidas en el archivo de configuración `SpanishConfig.json`, como se muestra en la Figura 5.2. Las oraciones se generaron tomando en consideración la información requerida para la atención de dicha petición. En el caso de “Agendar una cita médica”, los datos a obtener son: 1) nombre del médico, 2) hora de la cita y 3) día de la cita. Para la petición de “Recordatorio de toma de medicamento”, los datos solicitados son: 1) fecha y 2) hora. Finalmente, para la “Búsqueda de información”, se toma en cuenta el texto que se encuentra después de alguna palabra que refiera a la búsqueda de algo (e.g., en la oración “Busca información sobre la Diabetes”, la palabra “Buscar” indica el tipo de acción y se toma como elemento a enviar al *Broker Service* el texto posterior a dicha palabra).

```

"SENTENCES":{
  "APPOINTMENT":{
    "appointment1":"Agendar una cita para el dia jueves a las 7 y media de la mañana con el doctor Martinez",
    "appointment2":"cita para mañana con el doctor Ivan Valdespin a las tres y cuarto de la tarde",
    "appointment3":"hazme una cita con el medico temprano en 15 de enero"
  },
  "REMINDERS":{
    "reminder1":"Recuerdame tomar mi medicina el dia de mañana a las 8 y cuarto de la mañana",
    "reminder2":"acuerdame que tengo que tomar mi medicina hoy a las 7:20",
    "reminder3":"recordar tomar mi medicamento a las 7 de la noche"
  },
  "SEARCH":{
    "search1":"Busca informacion sobre la diabetes",
    "search2":"dime como debo tomar mi medicamento para la diabetes",
    "search3":"¿como se toman las pastillas de paracetamol?",
    "search4":"buscame informacion sobre las personas con diabetes"
  }
},

```

Figura 5.2: Oraciones utilizadas para identificación del tipo de peticiones

Se realizaron varias pruebas para la detección del tipo de petición con base en la similitud semántica. Se inicio definiendo cinco oraciones por cada tipo de petición y el procedimiento a seguir es el siguiente. Primero se obtiene la similaridad de la petición generada por el usuario por cada oración predefinida y se calcula el promedio de la similaridad de cada tipo de petición, generando así tres promedios de similaridad (uno por cada tipo de petición). Se tomó en consideración el promedio más grande para la identificación del tipo de petición. Sin embargo, esto generó dos problemas: 1) el tiempo de cómputo era considerablemente alto (aproximadamente tres segundos) y 2) la identificación del tipo de petición no era precisa, debido a que el promedio obtenido por cada petición podía variar de manera significativa, si alguna de las oraciones obtenía un valor muy bajo, ocasionando que, a pesar de que las demás oraciones obtenían valores altos, el promedio se veía reducido por una similaridad baja.

Finalmente, después de varias pruebas se definieron tres oraciones para la peticiones de “Agendar cita médica” y “Recordatorio de medicamento”, y cuatro oraciones para la petición de “Buscar Información”, esto último debido a la gran variedad de maneras que existen para expresar la acción de buscar información. Así mismo, se omitió el cálculo de promedios de similaridad para evitar los errores mencionados, y en consecuencia, se toma en consideración la similaridad más alta de las diez generadas. Por otra parte, se definió un umbral de similaridad igual a 0.7, con el objetivo de identificar cuándo una petición no sea referente a los tipos de peticiones que se pueden atender en este caso de estudio. La implementación de estos parámetros mejoró la precisión del sistema. Los resultados obtenidos pueden ser consultados en la Sección 6.1.1 del Capítulo 6.

Para la identificación de la información requerida se generaron patrones, bajo el concepto de coincidencias basadas en reglas, como se menciona en la Sección 4.3.7 del Capítulo 4. Estos patrones se encuentran contenidos en la clase `SpanishConfig` dentro del método `createPatterns`. Se generaron un total de 24 patrones encargados de la identificación de información necesaria para el envío de la información al *Broker Service* del compañero digital. Algunos de los patrones generados pueden visualizarse en la Figura 5.3.

```
def createPatterns(self):

    #Patterns for finding the physician in the sentences
    physician_pattern1 = [{'LOWER':'doctor','OP':'+'},{'POS':'PROPN','OP':'+'}] #Doctor PROPNOUN
    physician_pattern2 = [{'LOWER':'doctora','OP':'+'},{'POS':'PROPN','OP':'+'}]#Doctora PROPNOUN

    #Patterns for finding dates
    date_pattern1 = [{'POS':'NUM'},{'POS':'ADP'},{'POS':'NOUN'}] #DIA de MES
    #date_pattern2 = [{'LOWER':'mañana'}]
    date_pattern2 = [{'LOWER':'lunes','OP':'?'},{'LOWER':'martes','OP':'?'},
                    {'LOWER':'miercoles','OP':'?'},{'LOWER':'miércoles','OP':'?'},
                    {'LOWER':'jueves','OP':'?'},{'LOWER':'viernes','OP':'?'},
                    {'LOWER':'sábado','OP':'?'}, {'LOWER':'sabado','OP':'?'}] #Dias de la semana

    date_pattern3 = [{'LOWER':'la','OP':'!'},{'LOWER':'mañana','OP':'+'}]

    #Patterns for finding time in an appointment

    time_pattern1 = [{"POS":"NUM","OP":"+"},{"POS":"CCONJ","OP":"+"},{"LOWER":"media","OP":"+"},
                    {'LOWER':'de','OP':'?'},{'LOWER':'la','OP':'?'},{'LOWER':'mañana','OP':'?'},
                    {'LOWER':'tarde','OP':'?'},{'LOWER':'noche','OP':'?'}] #numero y media (opc)

    time_pattern2 = [{"POS":"NUM","OP":"+"},{"POS":"CCONJ","OP":"+"},{"LOWER":"cuarto","OP":"+"},
                    {'LOWER':'de','OP':'?'},{'LOWER':'la','OP':'?'},{'LOWER':'mañana','OP':'?'},
                    {'LOWER':'tarde','OP':'?'},{'LOWER':'noche','OP':'?'}] #numero y cuarto (opc)
```

Figura 5.3: Patrones diseñados para la extracción de información relevante en español

Los patrones generados mediante el método `getFeatures` posibilitan la identificación de información necesaria en cada tipo de solicitud, una vez que esta ha sido reconocida mediante el método `getIntention` de la clase `RelevantInformation`. La implementación de estos patrones en el micro-servicio permite la identificación eficaz de la información requerida, incluso cuando se proporciona de diversas maneras. Esta capacidad confiere robustez al micro-servicio, ya que los usuarios no están obligados a formular sus solicitudes de manera estandarizada (e.g., especificando la fecha de múltiples formas, como el día y mes, e incluso usando expresiones como “mañana”).

Las maneras en que el usuario puede proporcionar la información requerida son las siguientes:

- Fecha:
 1. Día de la semana (de Lunes a Sábado)

2. Expresiones como: mañana
 3. Día-mes (5 de octubre)
 4. Día-mes-año (5 de octubre del 2023)
- Hora:
 1. Horario del día (en la mañana, tarde o noche)
 2. Expresión como: "Temprano"
 3. Hora-minutos (3:28)
 4. Hora-minutos-horario del día (3:28 de la mañana, tarde o noche)
 5. Hora-Expresión (tres y media, tres y cuarto)
 - Médico:
 1. Doctor(a)-Nombre del doctor(a)
 2. Médico-Nombre del médico

Esta capacidad de detectar información de múltiples formas contribuye a la naturalidad en la interacción del usuario con el sistema, proporcionando una experiencia más cómoda. La flexibilidad del micro-servicio para comprender variaciones en las solicitudes hace que los usuarios se sientan más a gusto al utilizar la plataforma. Se puede observar un ejemplo de estas variaciones en la Figura 5.4. Además, esta adaptabilidad del sistema a expresiones más informales o diversificadas optimiza la usabilidad y eficacia del micro-servicio.

```
Raw: Recuerda me tomar mi medicamento mañana en la 7 y media de la noche.  
Action: Recordatorio de toma de medicamento  
Place:  
Physician:  
Date: mañana  
Time: 7 y media de la noche
```

Figura 5.4: Prueba de extracción de características de recordatorio de medicamento

Como se puede observar en la Figura 5.4, al obtener la información, esta se organiza siguiendo un formato *json* especificado en el archivo de configuración `SpanishConfig.json`. Este archivo, como se observa en la Figura 5.5, está constituido por diferentes elementos, ya que el *Broker Service* del compañero digital pretende atender diversas peticiones basadas en el contexto médico. Sin embargo, para este caso de estudio, sólo se utilizan los siguientes elementos:

```
"BROKERFORMAT":{  
  "raw":"","  
  "action":"","  
  "search":"","  
  "place":"","  
  "physician":"","  
  "date":"","  
  "time":"","  
  "timestamp":"","  
  "user-id":"","  
  "issue":"","  
},
```

Figura 5.5: Formato *json* para la interacción con el *Broker Service*

- *raw*: contiene la petición traducida del usuario.
- *action*: contiene el tipo de petición a realizar.
- *search*: contiene la información a buscar en caso de una solicitud de búsqueda.
- *physician*: contiene el nombre del médico.
- *date*: contiene la fecha en caso de solicitudes de agendar cita o recordatorios.
- *time*: contiene el horario en el caso de recordatorios o agendar citas.
- *timestamp*: contiene la hora en que se generó la petición.

Finalmente, una vez que se ha generado el formato *json* con la información requerida para la atención de la petición por el compañero digital, se utiliza el método `Producer` de la clase `BrokerCommunication` para el envío de la información mediante el *framework* `Kafka`. Se realizaron pruebas del envío de la información contenida en el formato *json* a un *Broker Service* implementado de manera local, con el fin de comprobar que el sistema de reconocimiento de voz y habla multi-fuente es capaz de funcionar de manera adecuada para este caso de estudio. La Figura 5.6 muestra los resultados obtenidos de una de las pruebas realizadas, en donde se demuestra el correcto envío de la información requerida para la atención de la petición de un usuario.

```

[2023-06-29 12:16:44,622] INFO [Controller id=1] Process
ing automatic preferred replica leader election (kafka.controller.KafkaControl
ler)
descargas-kafka-1 | [2023-06-29 12:16:44,622] TRACE [Controller id=1] Check
ing need to trigger auto leader balancing (kafka.controller.KafkaController)
descargas-kafka-1 | [2023-06-29 12:16:44,627] DEBUG [Controller id=1] Topic
s not in preferred replica for broker 1 HashMap() (kafka.controller.KafkaControl
ler)
descargas-kafka-1 | [2023-06-29 12:16:44,627] TRACE [Controller id=1] Leader
balance ratio for broker 1 is 0.0 (kafka.controller.KafkaController)
descargas-kafka-1 | [2023-06-29 12:21:44,627] INFO [Controller id=1] Process
ing automatic preferred replica leader election (kafka.controller.KafkaControl
ler)
descargas-kafka-1 | [2023-06-29 12:21:44,627] TRACE [Controller id=1] Check
ing need to trigger auto leader balancing (kafka.controller.KafkaController)
descargas-kafka-1 | [2023-06-29 12:21:44,630] DEBUG [Controller id=1] Topic
s not in preferred replica for broker 1 HashMap() (kafka.controller.KafkaControl
ler)
descargas-kafka-1 | [2023-06-29 12:21:44,631] TRACE [Controller id=1] Leader
balance ratio for broker 1 is 0.0 (kafka.controller.KafkaController)

```

```

ivan@valdespin: ~/kafka/bin
{"raw": "Cita para ma\u00f1ana a las 4.40 de la tarde con la doctora Laura.", "a
ction": "Agendar cita", "place": "", "physician": "doctora Laura", "date": "ma\u
00f1ana", "time": "4:40 de la tarde", "timestamp": "14:24:21", "user-id": "", "i
ssue": "1"}
{"raw": "Agenda cita con la doctora Irma el viernes temprano.", "action": "Agen
dar cita", "place": "", "physician": "doctora Irma", "date": "viernes", "time": "
temprano", "timestamp": "14:24:22", "user-id": "", "issue": "1"}
{"raw": "Necesito una cita para ma\u00f1ana con el doctor Juan a las ocho y medi
a de la ma\u00f1ana.", "action": "Agendar cita", "place": "", "physician": "doct
or Juan", "date": "ma\u00f1ana", "time": "ocho y media de la ma\u00f1ana", "time
stamp": "14:24:22", "user-id": "", "issue": "1"}
{"raw": "Hasme una cita a las 3.20 de la tarde con el doctor ivan el 23 de enero
.", "action": "Agendar cita", "place": "", "physician": "Dr./Dra. ivan", "date":
"23 de enero", "time": "2:3 de", "timestamp": "14:24:22", "user-id": "", "issue
": "1"}
{"raw": "Dame una cita para el jueves con el doctor Maldonado.", "action": "Agen
dar cita", "place": "", "physician": "doctor Maldonado", "date": "jueves", "time
": "", "timestamp": "14:24:22", "user-id": "", "issue": "1"}
{"raw": "Agenda para ma\u00f1ana temprano una cita con el doctor Carmona porfavo
r.", "action": "Agendar cita", "place": "", "physician": "doctor Carmona porfavo
r", "date": "ma\u00f1ana", "time": "temprano", "timestamp": "14:24:22", "user-id
": "", "issue": "1"}

```

(a) Inicialización del *Broker Service* de manera local utilizando el *framework Kafka*

(b) Información recibida por el *Broker Service*, proveniente del micro-servicio

Figura 5.6: Inicialización y recibimiento de información por parte del *broker* implementado de manera local, utilizando el *framework Kafka*

5.2. Sistema para la interacción con dispositivos de Internet de las Cosas

A continuación, se aborda el segundo caso de estudio para el sistema de reconocimiento de voz y habla multi-fuente. En la Sección 5.2.1 se realiza una descripción general del caso de estudio, grupo de la Sección 5.2.2 en donde se describe la adaptación realizada al sistema de reconocimiento de voz y habla multi-fuente para abordar este caso de estudio.

5.2.1. Descripción general

El segundo caso de estudio se inscribe en un proyecto desarrollado por la Universidad de St. Gallen, Suiza, el cual tiene como objetivo utilizar el sistema de reconocimiento de voz y habla multi-fuente para facilitar la interacción de los usuarios con dispositivos de Internet de las Cosas, siguiendo una estructura basada en *Thing Description* [WoT Working Group, 2023].

Como escenario de prueba se realizó la interacción con dispositivos que se utilizan dentro de un ámbito industrial. Por esta razón, los dispositivos propuestos para la interacción con el usuario son: 1) brazo robot, 2) lámpara RGB y 3) persianas automatizadas. En dicho escenario la identificación de usuarios no es requerida, ya que se pretende que cualquier persona pueda interactuar con los dispositivos mencionados.

Las peticiones que el usuario puede solicitar, en el idioma inglés, mediante el sistema de reconocimiento de voz y habla para la interacción con dispositivos de Internet de las Cosas son:

- Interacción con el brazo robot
 1. Información sobre el estado del robot (libre u ocupado)
 2. Inicialización del robot
 3. Movimiento del robot en los ejes X,Y y Z
 4. Apertura y cerradura del *gripper* del robot
 5. Finalización de la interacción con el robot

- Interacción con la lámpara RGB
 1. Encendido y apagado de la lámpara
 2. Incremento y decremento del brillo de la lámpara
 3. Cambio del color de la lámpara

- Interacción con las persianas automáticas
 1. Subir las persianas
 2. Bajar las persianas
 3. Parar las persianas

5.2.2. Identificación de peticiones y extracción de información para la comunicación con dispositivos del Internet de las Cosas

Como se detalla en la Sección 4.3.7, la extracción de información relevante se lleva a cabo mediante el empleo de clases diseñadas para generar patrones, utilizando el concepto de coincidencias basadas en reglas y adaptándose al idioma utilizado. En el contexto de este estudio, la clase `EnglishConfig` es la seleccionada para este propósito.

En cuanto a la identificación del dispositivo que el usuario desea utilizar, también se emplean patrones. Se han creado tres patrones específicos, lo cuales se ilustran en la Figura 5.7, para identificar las palabras asociadas a cada dispositivo. Estas palabras incluyen sinónimos que pueden ser utilizados para referirse a cada dispositivo en particular. Para el caso de la lámpara, las palabras identificadoras son: *lamp* y *lights*. Para el robot, las palabras asociadas son: *robot* y *cherrybot*. Finalmente, para las persianas, se utilizan las palabras *blinds* y *curtains*.

```

createPatterns(self):

objects = self.english_objects['OBJECTS']

lamp_synonyms = [{"LOWER":{"FUZZY1":{"IN":objects['LAMP']['synonyms']}},'OP': '+'}]
blind_synonyms = [{"LOWER":{"FUZZY1":{"IN":objects['BLINDS']['synonyms']}},'OP': '+'}]
robot_synonyms = [{"LOWER":{"FUZZY1":{"IN":objects['ROBOT']['synonyms']}},'OP': '+'}]

```

Figura 5.7: Patrones generados para la identificación del dispositivo a utilizar

Como se puede observar en la Figura 5.7, se hace uso de función `FUZZY1`, que permite darle cierta flexibilidad al patrón que se genera para identificar palabras que sean ligeramente similares a las que se buscan. Existen tres tipos de funciones disponibles, `FUZZY1`, `FUZZY2` y `FUZZY3`. El incremento en el número de la función conlleva a una mayor amplitud en la asociación de palabras similares a la búsqueda realizada. En este escenario específico, se ha optado por `FUZZY1` con el objetivo de conferir una ligera flexibilidad al patrón generado.

La elección de `FUZZY1` busca permitir la detección de la palabra, incluso en casos donde la traducción de audio a texto no sea completamente precisa. La flexibilidad que brinda esta función se traduce en la capacidad de adaptarse a variaciones de una o dos letras, dependiendo de la longitud de la palabra. La decisión de no emplear las otras funciones se fundamenta en la consideración de que estas podrían generar un mayor índice de error, al permitir la asociación con múltiples palabras que presenten similitud con la búsqueda original.

Por otra parte, la lista de palabras identificadoras de los dispositivos, junto con las palabras utilizadas en la generación de patrones para la extracción de información, se encuentran detalladas en el archivo de configuración `EnglishConfig.json`. Cada dispositivo tiene su sección correspondiente en el formato `json` y las palabras clave están organizadas, dentro de una lista, para facilitar su uso en la generación de patrones. Este aspecto se ilustra en la Figura 5.8.

Como se puede observar en la Figura 5.8, cada dispositivo cuenta con un diccionario de palabras que ayudan a la generación de los patrones encargados de la identificación de la acción a realizar, dependiendo del dispositivo y, con base en esto, la información requerida.

```

"OBJECTS":{
  "LAMP":{
    "synonyms":["lamp","lights","brightness","color"],
    "properties":["on","off"],
    "actions":{
      "turnon":["enable","start","illuminate","activate"],
      "turnoff":["desable","deactivate","stop"],
      "color": ["change","switch"],
      "brightness_up": ["set","increase","rise","brighter"],
      "brightness_down":["lower","decrease","dim","reduce"]
    },
    "colors":["white","green","blue","pink","brown","yellow","red","purple"]
  },
  "BLINDS":{
    "synonyms": ["blind","blinds","curtains","shades"],
    "properties": ["up","down","stop"],
    "actions":{
      "up":["up","open","rise"],
      "down":["down","close","draw","drop","lower"],
      "stop":["stop","halt"]}
  },

```

Figura 5.8: Lista de palabras para la identificación de dispositivos y extracción de información relevante.

Dentro del método `createPatterns`, se generan siete patrones para identificar las acciones asociadas con la lámpara RGB, cuatro patrones para la interacción con las persianas automatizadas, trece patrones para la interacción con el brazo robot y un patrón responsable de la identificación de la palabra de activación para el uso del sistema de captura permanente, como se muestra en la Figura 5.9.

```

def createPatterns(self):

    objects = self.english_objects['OBJECTS']

    lamp_synonyms = [{"LOWER":{"FUZZY1":{"IN":objects['LAMP']['synonyms']}},'OP':'+'}]
    lamp_on = [{"LOWER":{"FUZZY1":{"IN":objects['LAMP']['actions']['turnon']}},'OP':'+'}]
    lamp_off = [{"LOWER":{"FUZZY1":{"IN":objects['LAMP']['actions']['turnoff']}},'OP':'+'}]
    colors = [{"LOWER":{"IN":objects['LAMP']['colors']}},'OP':'+'}]
    color_verbs = [{"LOWER":{"FUZZY1":{"IN":objects['LAMP']['actions']['color']}},'OP':'?'}]
    rise_brightness = [{"LOWER":{"FUZZY1":{"IN":objects['LAMP']['actions']['brightness_up']}},'OP':'+'}]
    reduce_brightness = [{"LOWER":{"FUZZY1":{"IN":objects['LAMP']['actions']['brightness_down']}},'OP':'+'}]

    blind_synonyms = [{"LOWER":{"FUZZY1":{"IN":objects['BLINDS']['synonyms']}},'OP':'+'}]
    blind_actions_up = [{"LOWER":{"FUZZY1":{"IN":objects['BLINDS']['actions']['up']}},'OP':'+'}]
    blind_actions_down = [{"LOWER":{"FUZZY1":{"IN":objects['BLINDS']['actions']['down']}},'OP':'+'}]
    blind_actions_stop = [{"LOWER":{"FUZZY1":{"IN":objects['BLINDS']['actions']['stop']}},'OP':'+'}]

    robot_synonyms = [{"LOWER":{"FUZZY1":{"IN":objects['ROBOT']['synonyms']}},'OP':'+'}]
    robot_status = [{"LOWER":{"FUZZY1":{"IN":objects['ROBOT']['actions']['status']}},'OP':'+'}]
    robot_start = [{"LOWER":{"FUZZY1":{"IN":objects['ROBOT']['actions']['actions'][4:6]}},'OP':'+'}]
    robot_movements = [{"LOWER":{"FUZZY1":{"IN":objects['ROBOT']['actions']['actions'][0:3]}},'OP':'+'}]
    robot_direction = [{"LOWER":{"FUZZY1":{"IN":objects['ROBOT']['properties']['movement']}},'OP':'+'}]
    robot_disconnect = [{"LOWER":{"FUZZY1":objects['ROBOT']['actions']['actions'][6]}},'OP':'+'}]
    robot_reset = [{"LOWER":{"FUZZY1":objects['ROBOT']['actions']['actions'][3]}},'OP':'+'}]
    gripper_movements = [{"LOWER":{"FUZZY1":{"IN":objects['ROBOT']['actions']['gripper']}},'OP':'+'}]

```

Figura 5.9: Patrones diseñados para la extracción de información relevante

El archivo `Objects.py` aloja tres clases: `Lamp`, `Blind`, y `Robot`, cada una diseñada para proporcionar métodos específicos que facilitan la identificación y ejecución de acciones asociadas a sus respectivos dispositivos y, como se menciona en la Sección 4.3.7 del Capítulo 3, hacen uso de la interfaz `IDevice`

Dentro de la clase `Lamp`, se incluyen métodos clave como `getStatus`, `getOnOff`, `getColor`, `getBrightness`, y `getmatches`. Este último está presente en todas las clases para identificar patrones detectados en la instrucción del usuario. La clase `Robot` se enfoca en las peticiones definidas en la sección correspondiente de este caso de estudio, con métodos como `startRobot`, `disconnectRobot`, `status`, `initialize`, `open/closeGripper`, `getCoordinates`, `moveRobot`, etc. Finalmente, la clase `Blind` incorpora los métodos `getUp`, `getDown`, y `getStop`, que se refieren a las acciones ejecutables por este dispositivo.

Para la generación del formato `json` correspondiente a cada dispositivo, se utilizan los formatos contenidos en el archivo `EnglishConfig.json`. Un ejemplo de estos se puede visualizar en la Figura 4.34a. Estos formatos van siendo llenado conforme se va obteniendo la información solicitada, mediante el método `getFeatures`.

Una vez que se ha generado el formato `json` correspondiente, se crea una instancia de la clase `IoTCommunication` y se utiliza uno de sus métodos correspondientes al tipo de acción a realizar (e.g., para inicializar el robot se hace uso del método `PUT`), ingresándole como

parámetro de entrada a dicho método, el formato *json* previamente generado, seguido del *endpoint* relacionado con el dispositivo y la acción a realizar. Finalmente, se cuenta con una serie de mensajes predeterminados, localizados en el archivo `EnglishConfig.json`, como se muestra en la Figura 5.10, que se despliegan al usuario, dependiendo de la respuesta obtenida por el dispositivo.

Si se recibe como respuesta un código 200, indicando que todo salió bien, dependiendo de la acción solicitada, se envía un mensaje al usuario indicándole que su petición ha sido procesada correctamente (e.g., si se recibe un código 200 a la acción “inicializar robot”, el mensaje mostrado al usuario es “*Robot successfully initialized. You are able to use it*”). Por otro lado, si se recibe un código 401, el mensaje mostrado es negativo (e.g., “*Robot is not free. You are not able to use it. Try later*”).

```

"messages":{
  "taken":"\n\n\nRobot isn't free.\n\n\nYou are not able to use it, try later",
  "free":"\n\n\nRobot is free.\n\n\nYou can use it",
  "error":"\n\n\nBad request.\n\n\nPlease try againg",
  "bad_parameter": "\n\n\n Bad input parameter. \n\n\n Try again.",
  "init_succes": "\n\n\n Robot Successfully initialized. \n\n\n You are able to use it",
  "max_distance_gripper": "\n\n\n Gripper is at its maximum opening.",
  "min_gripper_distance":"\n\n\n Gripper is totally closed.",
  "distance_exceeded": "\n\n\n Bad input.\n\n\n Parameters had been exceeded.",
  "bad_gripper_get": "\n\n\n Getting gripper data was wrong.",
  "get_error":"\n\n\n Getting robot coordenates was wrong",
  "not_distance":"\n\n\n Distance not specified",
  "not_direction":"\n\n\n Direction not specified",
  "success":"ok",
  "fail":"\n\n\nRequest not identified.\n\n\n Please try again",
  "bad_token":"\n\n\n Invalid token",
  "delete_user":"\n\n\n User Successfully desconected",
  "issue":"\n\n\n\n Something was wrong. \n\n\n Is not possible to desconect",
  "not_reset":"\n\n\n\n Robot can't move to start position"
}

```

Figura 5.10: Mensajes preestablecidos para la interacción con el usuario

5.3. Sistema de identificación de objetos basada en contexto

A continuación, se aborda el tercer caso de estudio para el sistema de reconocimiento de voz y habla multi-fuente. En la Sección 5.3.1 se realiza una descripción general del caso de estudio, seguido de la Sección 5.3.2 en donde se describe la adaptación realizada al sistema de reconocimiento de voz y habla multi-fuente para abordar este caso de estudio.

5.3.1. Descripción general

El tercer caso de estudio se inscribe en un proyecto desarrollado por la Universidad de St. Gallen, Suiza, el cual lleva por nombre *Object Identification using Contextual Information* (OIC).

Este sistema es una propuesta de compañero digital conformado por cuatro componentes basados en el ciclo de decisión que llevan a cabo los agentes clásicos [Weiss, 1999]. Los componentes que se proponen son *Percibir*, *Interpretar*, *Decidir* e *Interactuar*. OIC es un sistema enfocado en el entendimiento de escenas temporales para la asistencia de usuarios, basado en el comportamiento de los humanos cuando se les pide explicar qué es lo que sucede en una escena o video. Por lo cual, para el funcionamiento del sistema, se requiere de la adquisición de vídeo para su posterior procesamiento.

Los cuatro componentes mencionados están conformados por *Scene Graph Generator*, *Frame Graph Generator*, *Temporal Graph Builder* y *Large Language Model* (LLM), respectivamente. Estos cuatro elementos cumplen con un papel fundamental para el correcto funcionamiento del sistema. Algunas de sus funciones son:

- *Scene Graph*: recibe imágenes de una fuente de vídeo, como entrada, y genera un grafo de escena que representa objetos identificados y sus relaciones en la escena.
- *Frame Graph Generator*: toma el grafo de escena generado por el *Scene Graph Generator* y agrega identificadores únicos a las instancias de objetos del mismo tipo. Esto da como resultado un grafo de cuadro que se utiliza para realizar un seguimiento de los objetos en una escena.
- *Temporal Graph Builder*: integra los gráficos de cuadro en un gráfico temporal, que es una representación sensible al tiempo de los objetos rastreados a lo largo del tiempo. Permite el seguimiento de objetos en movimiento a medida que cambian de posición en el tiempo.
- *Large Language Module* (LLM): el LLM pre-entrenado se utiliza para enriquecer el *Temporal Graph Builder* con conocimiento externo y ajustar su rendimiento. El modelo es capaz de responder preguntas formuladas por un usuario, lo que implica que puede comprender y generar texto en lenguaje natural basado en la información del gráfico temporal.

Dentro de las limitaciones de este sistema, se encuentran la falta de etiquetas de relaciones en los conjuntos de datos. Esto significa que el sistema actualmente cuenta con una identificación de objetos limitada por parte del *Scene Graph*, haciendo que no se puedan generar relaciones de objetos más complejas y, por ende, se limita la capacidad de entendimiento del sistema. La confusión de escenarios con entidades u objetos similares, al ser

un sistema que obtiene toda su información mediante visión, carece de la información de audio que se encuentra en las escenas de vídeo, haciendo que en determinadas ocasiones el sistema tienda a realizar suposiciones sobre preguntas que requieren más información que la obtenida visualmente.

Se realizó la integración del sistema de reconocimiento de voz y habla al sistema OIC. Al ser un sistema de captura de audio multi-fuente capaz de realizar la traducción de audio a texto, esto le permite al sistema OIC la adquisición de diferentes fuentes de información (visual y audio) que ayudarían a mejorar algunas de las deficiencias mencionadas y, por ende, una mejor comprensión de los escenarios en donde se utilice el sistema, permitiéndole mejorar la calidad de las respuestas. Las acciones a realizar por parte del sistema de reconocimiento de voz y habla son:

- Captura de audio de manera constante durante el desarrollo de la escena a evaluar.
- Traducción de la información de audio capturada por los sistemas de captura permanente a un formato de texto.
- Integración de la información en formato de texto al archivo generado por el sistema OIC, previo a su procesamiento utilizando LLM.

Una vez realizadas dichas acciones, se deberá realizar una comparativa de las respuestas generadas por el sistema OIC con y sin la información capturada y procesada por el sistema de reconocimiento de voz y habla.

5.3.2. Integración del sistema de reconocimiento de voz y habla multi-fuente con el sistema OIC

El proceso de operación que se lleva a cabo en la integración del sistema de reconocimiento de voz y habla multi-fuente con el sistema OIC es el siguiente:

Para la integración de ambos sistemas, solamente se utiliza como fuente de entrada al sistema de captura permanente, ya que se requiere la grabación constante de audio durante algún escenario de prueba. Por esta razón, la aplicación Android no es utilizada en este caso de estudio. El sistema de captura permanente, cuenta con una variable definida en su archivo de configuración, referente al número de *buffers* que el sistema debe generar y enviar al micro-servicio. La definición de esta variable, para fines de prueba de la integración de ambos sistemas, es igual al tiempo aproximado en que se lleva a cabo el escenario de prueba, dividido entre los segundos de audio que el *buffer* es capaz de almacenar.

El funcionamiento del sistema de captura permanente cambia ligeramente para este caso de estudio, ya que sigue utilizando la palabra de activación para comenzar a grabar. Sin embargo, ahora en vez de generar y enviar sólo un *buffer* que contiene la petición del usuario y regresar al estado de escucha de la palabra de activación, se requiere el envío de múltiples *buffers* de audio de cinco segundos, de manera constante, posterior a la detección de la palabra de activación.

Estos *buffers* de audio son enviados a un *endpoint* del micro-servicio mediante el método POST, destinado a este caso de estudio para la generación de un solo archivo de audio que contenga, de manera continua, la grabación de audio del escenario de prueba. Para el envío de cada *buffer* de audio al micro-servicio, la petición contiene dos *headers* que permiten al micro-servicio identificar el número de audio enviado y el dispositivo de donde proviene. Estos *headers* llevan por nombre **Audio** y **Disp**, respectivamente.

La identificación de los dispositivos permite la integración futura de más de un sistema de captura permanente en los escenarios y ayuda al micro-servicio a generar un repositorio por cada uno de ellos, evitando la mezcla de la información recibida por cada dispositivo de captura permanente. Por otro lado, la identificación de cada *buffer* de audio enviado, permite la generación de los archivos de audio de manera secuencial, evitando con esto que, al momento de unir todos los archivos de audio, se unan de manera mezclada, generando un archivo de audio completo erróneo. La definición de los *headers* mencionados se pueden observar en la Figura 5.11.

```
//Headers
http.addHeader("Content-Type", CONTENT_TYPE);
if (!flag_task){
    audio_number += 1;
    String number = String(audio_number);
    http.addHeader("Disp", "Disp1");
    http.addHeader("Audio", number);
}
```

Figura 5.11: *Headers* establecidos para el envío de *buffers* de audio

EL micro-servicio cuenta con dos *endpoints* destinados a la integración con el sistema OIC. El primer *endpoint* denominado **Audio**, mostrado en la Figura 5.12, puede ser utilizado realizando una petición mediante el método POST. El objetivo de este *endpoint* es recibir los *buffers* de audio de los sistemas de captura permanente, procesarlos y generar los repositorios correspondientes para guardar los archivos de audio generados.

```

@app.route("/Audio", methods=["POST"])
def esp32audio():

    audio_data = request.data
    audio_data_array = np.frombuffer(audio_data, dtype=np.int16)
    device_id = request.headers.get('Disp')
    req = request.headers.get('Audio')
    AudioFile().oicFiles(req,device_id,audio_data_array)

    return "OK"

```

Figura 5.12: *Endpoint* “Audio” para la captura y procesamiento de *buffers* de audio relacionados con el sistema OIC

Para el procesamiento de los *buffers* de audio recibidos, se utiliza la función `frombuffer` de la biblioteca Numpy que convierte la información recibida en un *array*. Posteriormente, se obtienen los *headers* de identificación para la generación del archivo de audio en formato `.wav` y los repositorios correspondientes para almacenarlos. Esto se realiza utilizando la función `headers.get` de la biblioteca `request` y la clase `AudioFile`, a la cual se agregó un nuevo método denominado `oicFiles`, con el propósito de generar los archivos de audio y repositorios correspondientes de este caso de estudio. Este método genera un repositorio denominado OIC cuando se realiza la primera solicitud. Dentro de este repositorio se generan los repositorios que contienen los archivos de audio generados con los *buffers* enviados, correspondientes a los dispositivos de captura permanente que realizan solicitudes al micro-servicio, utilizando el *endpoint* Audio. Esto se puede visualizar en la Figura 5.13.

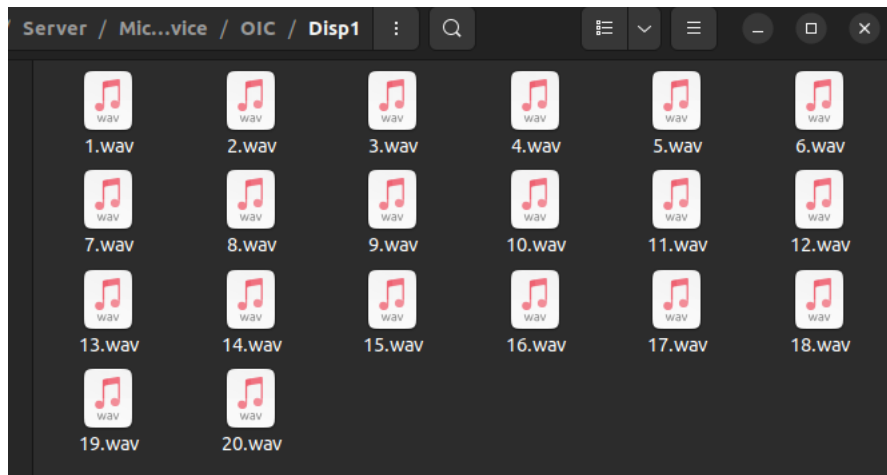


Figura 5.13: Generación de directorios y almacenamiento de archivos recibidos para la integración del micro-servicio con el sistema OIC.

El segundo *endpoint*, denominado OIC, puede ser utilizado realizando una petición mediante el método GET. Su propósito es enviar, en un formato de texto como respuesta al sistema OIC, la información obtenida por los sistemas de captura permanente. Por esta razón, el sistema OIC debe realizar una petición mediante el método GET, antes de implementar el modelo LLM, para unir la información obtenida de los sistemas de captura permanente con la información generada por el sistema OIC.

Una vez que el sistema OIC realiza la petición, el micro-servicio verifica si existe el repositorio OIC, ya que en caso contrario significaría que aún no se ha grabado el audio de algún escenario. Posteriormente, se verifica si existe el repositorio `TextFiles`, el cual contiene los archivos de texto generados de la traducción de los archivos de audio almacenados. Si ya existe algún archivo de texto generado anteriormente, se envía ese archivo de texto al sistema OIC para su integración con la demás información. En caso contrario, se verifica que exista algún repositorio o repositorios que contengan los archivos de audio obtenidos por los sistemas de captura permanente. Cabe recordar que el micro-servicio está acondicionado para la integración de más de un dispositivo de captura permanente en este caso de estudio. En caso de existir los archivos de audio, estos se unen mediante la función `AudioSegment.from_file` de la biblioteca `pydub`, generando un archivo de audio denominado `Disp''numerodedispositivo''Audio.wav`. Este audio se procesa utilizando la clase `FilterFactory` para aplicarle un filtro paso banda. Posteriormente, se utiliza el método `audioToText` para su traducción, generando un archivo de texto denominado `Disp''numerodedispositivo''.txt` con la información traducida, como se muestra en la Figura 5.14. Finalmente, este archivo es enviado al sistema OIC para su integración.

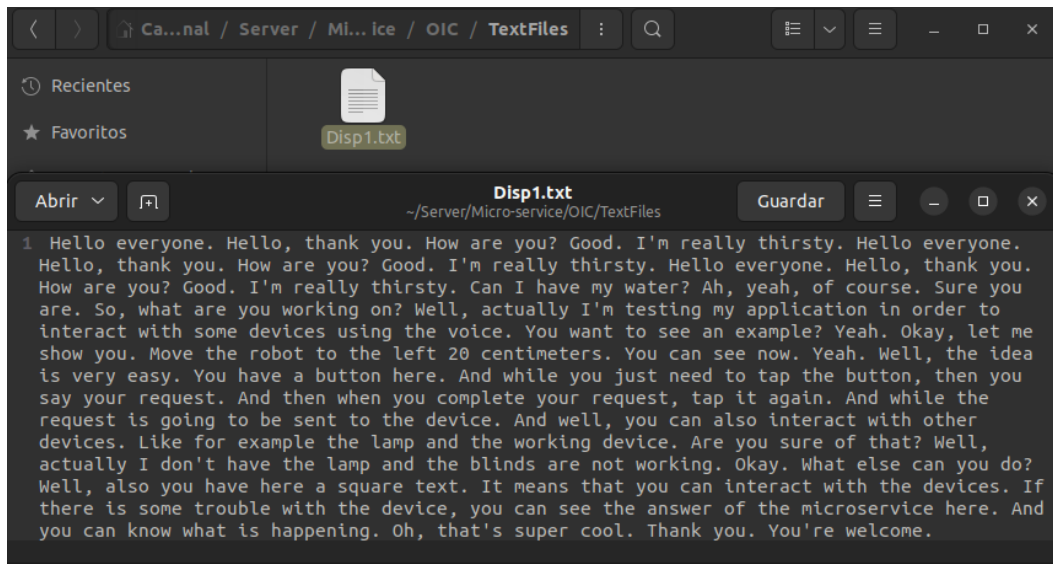


Figura 5.14: Generación del archivo de texto referente a la traducción del audio obtenido por sistemas de captura permanente

Como se observa en la Figura 5.14, los resultados obtenidos de la traducción de audio indican que **Whisper** no es capaz de separar los enunciados dichos por diferentes personas. A pesar de realizar una traducción del audio muy certera, la traducción se realiza de manera secuencial, siendo indiferente a qué persona corresponde cada enunciado. Esto genera algunos inconvenientes reportados en la Sección 6.3 del Capítulo 6. Sin embargo, a pesar de esto, los resultados obtenidos fueron considerablemente buenos.

Capítulo 6

Pruebas y resultados

En este capítulo se reportan las pruebas y resultados obtenidos de la evaluación del sistema de reconocimiento de voz y habla. En la Sección 4.2.5, se muestran las pruebas y resultados obtenidos referentes al rango de captura de audio del sistema de captura permanente. A continuación, en la Sección 6.1 se describen las pruebas y resultados referentes al primer caso de estudio, mientras que en la Sección 6.2 se exponen las pruebas y resultados de la evaluación del segundo caso de estudio. Finalmente, en la Sección 6.3, se detallan las pruebas realizadas para la evaluación del tercer caso de estudio.

6.1. Compañero digital para el cuidado de la diabetes

A continuación se presentan las pruebas realizadas para evaluar el sistema de reconocimiento de voz y habla correspondiente al primer caso de estudio. En la Sección 6.1.1 se muestran las pruebas y resultados de la identificación de peticiones. Después, en la Sección 6.1.2 se exponen los resultados logrados en la prueba de obtención de características. Finalmente, en la Sección 6.1.3, se muestran los resultados obtenidos en las pruebas sobre el tiempo de respuesta del sistema de reconocimiento de voz y habla.

6.1.1. Identificación de peticiones

Para poder evaluar la precisión, sensibilidad y exactitud del sistema de reconocimiento de voz y habla para la identificación del tipo de peticiones que es capaz de atender este caso de estudio (i.e., Agenda de citas con el médico, Recordatorio de toma de medicamento y Búsqueda de información referente a la diabetes), se elaboraron 15 oraciones de prueba por cada tipo de petición, siendo un total de 45 oraciones, tomando en cuenta un total de cinco personas, tres pacientes que padecen de diabetes y dos personas que no padecen de esta enfermedad.

Las oraciones elaboradas varían en el orden en que se menciona la información requerida para poder llevar a cabo la petición del usuario (e.g., para agendar una cita, la hora se menciona al principio, a la mitad o al final de la oración). Incluso se elaboraron algunas que omiten parte de la información requerida, con el fin de evaluar tanto la capacidad del sistema para detectar correctamente el tipo de petición como la información requerida.

Para poder obtener las métricas de evaluación mencionadas, se elaboró una matriz de confusión que puede visualizarse en la Tabla 6.1. Con base en esta matriz, se obtuvieron los primeros resultados de las métricas del sistema de reconocimiento de voz y habla mostrados en la Tabla 6.2

Tabla 6.1: Primera matriz de confusión para la evaluación de la identificación del tipo de peticiones.

	Cita médica	Recordatorio	Búsqueda
Cita médica	14	0	1
Recordatorio	2	13	0
Búsqueda	1	1	13

Tabla 6.2: Primeros resultados de evaluación del sistema de reconocimiento de voz y habla para la identificación de peticiones

Tipo de petición	Precisión	Sensibilidad	Exactitud
Cita médica	0.82	0.93	0.77
Recordatorio	0.92	0.86	0.81
Búsqueda	0.92	0.86	0.81

Los resultados obtenidos en las primeras pruebas indican que el sistema de reconocimiento de voz y habla cuenta con una alta precisión y sensibilidad, ya que los valores obtenidos para ambas métricas se encuentran por arriba del 0.8, correspondiente (en términos de porcentaje, a un 80%). Al tener una buena precisión y sensibilidad, esto indica que el sistema es capaz de identificar correctamente, en la mayoría de las veces, el tipo de petición que se le solicita.

Sin embargo, como se puede observar en la Tabla 6.1, cuando el sistema tiende a cometer errores en la identificación del tipo de petición, generalmente lo atribuye a la petición de “Agendar una cita con el médico” y, en su mayoría, esto sucede cuando se trata de peticiones de recordatorio de medicamentos. Esto genera que, para el tipo de petición “Agendar una cita con el médico”, la exactitud sea por debajo del 0.8, correspondiente al 80%.

Se realizó un análisis de las oraciones en donde el sistema de reconocimiento de voz y habla cometió errores al identificar el tipo de petición y se detectó que las oraciones mal identificadas eran las que carecían de información. Por ende, el sistema tiende a cometer errores cuando se proporciona muy poca información sobre la petición, ya que en el caso de los dos primeros tipos de peticiones (i.e., agendar cita y recordatorio de medicamento) es necesario proporcionar información similar como la hora y el día, de manera que si se omite la información que diferencia estas peticiones, el sistema de reconocimiento de voz y habla las confundirá.

Para corroborar lo antes mencionado, se agrego más información a las oraciones que fueron mal identificadas y se volvió a realizar la prueba de identificación utilizando las 45 oraciones. Los resultados obtenidos pueden visualizarse en las Tablas 6.3 y 6.4.

Tabla 6.3: Segunda matriz de confusión para la evaluación de la identificación del tipo de peticiones.

	Cita médica	Recordatorio	Búsqueda
Cita médica	15	0	0
Recordatorio	1	14	0
Búsqueda	0	0	15

Tabla 6.4: Segundos resultados de evaluación del sistema de reconocimiento de voz y habla para la identificación de peticiones

Tipo de petición	Precisión	Sensibilidad	Exactitud
Cita médica	0.93	1	0.93
Recordatorio	1	0.93	0.93
Búsqueda	1	1	1

Una vez que se agregó mayor información a las peticiones mal identificadas por el sistema de reconocimiento de voz y habla, se puede observar en la Tabla 6.3 que, a pesar de seguir teniendo equivocaciones, la identificación ha sido mejorada considerablemente, incrementando el valor obtenido en todas las métricas de evaluación, incluyendo la exactitud de la petición "Agendar cita médica" a un 0.93, correspondiente a un 93%. Esto indica que mientras mayor sea la información proporcionada por el usuario en su petición, el sistema de reconocimiento de voz y habla tiende a funcionar mejor. Por otro lado, si se proporciona sólo información que tienen en común los diferentes tipos de peticiones, aumenta la probabilidad de que el sistema de reconocimiento de voz y habla se equivoque.

6.1.2. Obtención de características

Una vez que se ha verificado que el reconocedor de voz y habla tiene una buena precisión al identificar el tipo de peticiones que se le hacen por parte de los usuarios, se realizaron pruebas para poder determinar con qué precisión, sensibilidad y exactitud el sistema es capaz de obtener las características necesarias para el envío de la información al *Broker Service* del compañero digital. Para la obtención de las métricas mencionadas, se utilizaron las 45 oraciones utilizadas para la prueba de identificación de peticiones.

Se generó una matriz de confusión por cada característica de importancia (Médico, Fecha, Hora y Buscar información) para la obtención de la precisión, sensibilidad y exactitud que tiene el sistema por cada característica. Los resultados de las matrices de confusión pueden observarse en las Tablas 6.5, 6.6, 6.7 y 6.8. Los resultados de precisión, sensibilidad y exactitud pueden observarse en la Tabla 6.9.

Tabla 6.5: Matriz de confusión de la característica “Médico”

	Verdad	Falso
Verdad	14	0
Falso	0	16

Tabla 6.6: Matriz de confusión de la característica “Fecha”

	Verdad	Falso
Verdad	27	0
Falso	0	3

Tabla 6.7: Matriz de confusión de la característica “Tiempo”

	Verdad	Falso
Verdad	24	3
Falso	1	2

Tabla 6.8: Matriz de confusión de la característica “Buscar información”

	Verdad	Falso
Verdad	12	0
Falso	3	0

Tabla 6.9: Resultados de evaluación del sistema de reconocimiento de voz y habla para la identificación de características

Características	Precisión	Sensibilidad	Exactitud
Médico	1	1	1
Fecha	1	1	1
Hora	0.88	0.95	0.86
Busqueda de información	1	0.8	0.8

Los resultados obtenidos muestran que el sistema de reconocimiento de voz y habla es capaz de identificar muy bien cuando se proporciona la información referente al médico y

a la fecha. Sin embargo, para la detección de horarios, se obtuvieron valores de precisión por debajo del 0.9. Esto se debe a las múltiples maneras en que las personas suelen definir un horario, ya que en ocasiones se utilizan expresiones como media y cuarto para referirse a 30 y 15 minutos, respectivamente. Sin embargo, el sistema de reconocimiento de voz y habla cuenta con valores superiores a 0.8 en las tres evaluaciones, por lo cual se puede considerar aceptable.

6.1.3. Tiempo de respuesta del sistema

Para la obtención promedio del tiempo de respuesta del sistema de reconocimiento de voz y habla, en este caso de estudio se realizaron 50 peticiones al micro-servicio por cada fuente de entrada (aplicación Android y sistema de captura permanente) y se midió el tiempo de respuesta. El tiempo que se tomó en consideración es desde el momento en que el usuario realiza su petición hasta el momento en que se envía la información requerida al *Broker Service*. Los resultados obtenidos se muestran en la Tabla 6.10.

Tabla 6.10: Comparativa del tiempo de respuesta entre la aplicación Android y el sistema de captura permanente

Dispositivo	Tiempo éxito [s]	Tiempo error [s]
Aplicación	1.99	1.26
ESP32-S3	3.3	2.61

Como se puede observar, en los tiempos de respuesta reportados en la Tabla 6.10, el tiempo que tarda el sistema de reconocimiento de voz y habla en recibir y procesar la petición del usuario es superior a un segundo. Esto se debe al tiempo que toma la identificación de la petición, ya que se tiene que hacer una comparativa de semejanza con cada tipo de petición que el micro-servicio es capaz de atender. Cabe recordar que el sistema de reconocimiento de voz y habla está conformado por las fuentes de entrada y el micro-servicio, el cual es encargado del procesamiento de las peticiones. Así mismo, se puede notar que para el caso en que se realiza una petición utilizando el sistema de captura permanente, el tiempo incrementa ya que para que un usuario pueda expresar su petición es necesario mencionar primero la palabra de activación, la cual toma un aproximado de 1.43 segundos en ser procesada e identificada.

Además, se puede observar que existe tiempo de error, el cual hace referencia al tiempo promedio que toma el micro-servicio en enviar una respuesta negativa a la fuente de entrada que haya sido utilizada, informándole al usuario que su petición no puede ser procesada, debido a falta de información o que no se ha podido identificar el tipo de petición. Como se observa, el tiempo es menor, lo cual se debe a que, al identificar algún error, el proceso que se está llevando a cabo queda suspendido.

6.2. Interacción con dispositivos del Internet de las Cosas

A continuación, se presentan las pruebas realizadas para la evaluación del sistema de reconocimiento de voz y habla para la interacción con dispositivos del Internet de las Cosas. En la Sección 6.2.1 se muestran las pruebas y resultados de la identificación de dispositivos. Finalmente, en la Sección 6.2.2, se exponen los resultados logrados sobre el tiempo de respuesta.

6.2.1. Identificación de dispositivos del Internet de las Cosas

Para la evaluación del sistema de reconocimiento de voz y habla, en el segundo caso de estudio se realizaron 80 peticiones por cada dispositivo generando, siendo un total de 240 solicitudes para la evaluación.

Para la obtención de los parámetros de evaluación (precisión, sensibilidad y exactitud) del sistema de reconocimiento de voz y habla para la detección de los dispositivos del Internet de las Cosas, se elaboró una matriz de confusión por cada dispositivo. Las peticiones se realizaron, haciendo uso de los sinónimos utilizados para la interacción con los dispositivos (e.g., para la interacción con la lámpara, se utilizaron los nombres *lamp* y *lights*).

Tabla 6.11: Matriz de confusión del dispositivo “Robot”

	Verdad	Falso
Verdad	38	0
Falso	2	40

Tabla 6.12: Matriz de confusión del dispositivo “Cortinas”

	Verdad	Falso
Verdad	36	2
Falso	4	38

Tabla 6.13: Matriz de confusión del dispositivo “Lámpara”

	Verdad	Falso
Verdad	36	3
Falso	4	37

Como se puede observar en las Tablas 6.11, 6.12 y 6.13, el dispositivo que mejor tiende a identificar el micro-servicio es el brazo robot ya que, como se observa en su matriz de confusión (cf. Tabla 6.11), solamente en dos ocasiones de 40 no se logró identificar que el usuario quería realizar una petición al brazo robot. Además, los resultados obtenidos con los dispositivos restantes (lámpara y cortinas) no fueron malos, ya que solamente cuatro

de 40 peticiones no lograron ser identificadas.

Las peticiones en donde no se logró identificar el dispositivo con el que se deseaba interactuar, el problema se debió en su mayoría a una mala pronunciación de la palabra referente al dispositivo, provocando que el módulo de procesamiento de lenguaje natural del micro-servicio no tradujera, de manera correcta, la petición del usuario (e.g., la palabra *lights* puede ser interpretada como *likes*).

Como se puede observar en la Tabla 6.14, la mayoría los resultados obtenidos en el cálculo de las métricas de desempeño para la detección de los dispositivos superan el 0.9, a excepción de la sensibilidad para la detección del dispositivo “Lámpara”. Como se mencionó anteriormente, esto se debe a que existen ocasiones en donde el sistema no logra traducir, de manera apropiada, debido a una mala pronunciación o la calidad del audio recibido. Sin embargo, con base en los resultados obtenidos, se puede decir que para la interacción con dispositivos de Internet de las Cosas, el sistema de reconocimiento de voz y habla es un sistema confiable, ya que logra capturar y procesar, en su mayoría, las peticiones del usuario de manera correcta.

Tabla 6.14: Resultados de la evaluación del sistema de reconocimiento de voz y habla para la identificación de características

Dispositivo	Precisión	Sensibilidad	Exactitud
Robot	1	0.95	0.97
Lámpara	0.94	0.90	0.92
Cortinas	1	0.87	0.91

6.2.2. Tiempo de respuesta del sistema

Se elaboraron 50 pruebas por cada dispositivo IoT para el cálculo del tiempo de respuesta del sistema de reconocimiento de voz y habla. Debido a que las cortinas automáticas no se encontraban en funcionamiento, sólo se realizaron pruebas con los dos dispositivos restantes (brazo robot y lámpara RGB). Para la realización de las pruebas, el tiempo que se tomó en consideración, para la evaluación del tiempo de respuesta, fue el tiempo transcurrido desde que el usuario realiza la petición mediante alguna de las fuentes de entrada (sistema de captura permanente o aplicación Android) hasta el momento de la ejecución de la petición del usuario por alguno de los dispositivos (brazo Robot o lámpara RGB).

Se elaboró una comparativa de los resultados entre las fuentes de entrada, con el objetivo de identificar cuál de las dos resultaba más eficiente en cuanto al tema de velocidad. Los resultados obtenidos se muestran en las Tablas 6.15, 6.16 y en la Figura 6.1 en donde se

pueden observar los dispositivos de Internet de la Cosas y dos columnas cuyos encabezados indican éxito y error. Estos encabezados hacen referencia al tiempo que tarda el sistema en realizar una petición de manera correcta (éxito) y cuánto tarda al momento de suceder algún inconveniente (error) utilizando cada una de las fuentes de entrada.

Tabla 6.15: Comparativa del tiempo de respuesta utilizando aplicación Android

Dispositivo	Éxito Tiempo [s]	Error Tiempo [s]
Lámpara	1.54	1.35
Robot	2.93	1.37

Tabla 6.16: Comparativa del tiempo de respuesta utilizando ESP32-S3

Dispositivo	Éxito Tiempo [s]	Error Tiempo [s]
Lámpara	2.88	2.16
Robot	3.64	2.34

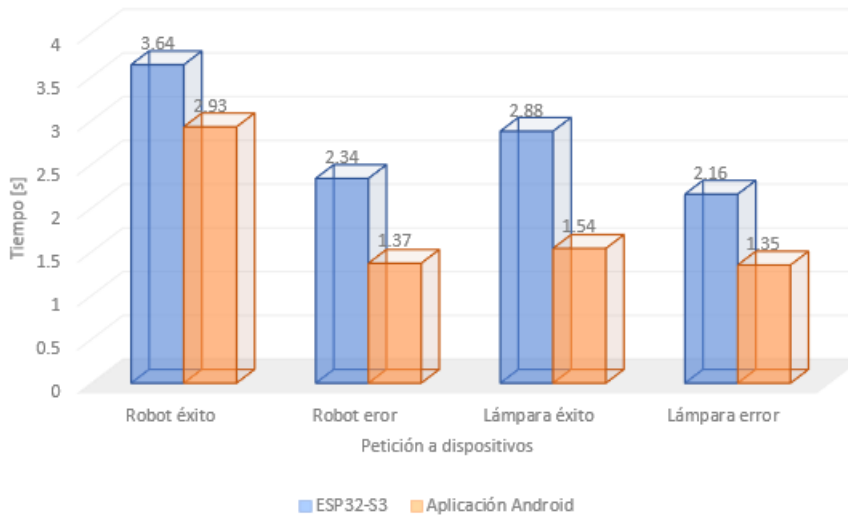


Figura 6.1: Comparativa del tiempo de interacción con dispositivos de Internet de las Cosas, utilizando diferentes fuentes de entrada

Como se muestra en la Figura 6.1, al comparar los resultados obtenidos cuando se utilizan ambas fuentes de entrada, resulta evidente que se logra una respuesta más rápida al utilizar la aplicación de Android para interactuar con cualquiera de los dispositivos de Internet de las Cosas, en comparación con el uso del sistema de captura permanente. Esto se debe a que, para enviar una solicitud utilizando el sistema de captura permanente, el primer paso es decir la palabra de activación para indicarle al sistema que se desea realizar una petición. Una vez que la palabra de activación ha sido identificada por el micro-servicio, este notifica al sistema de captura permanente indicándole que debe comenzar la grabación de la petición del usuario. Este proceso adicional resulta en un

tiempo de retardo de aproximadamente 1.43 segundos, como se observa en la Tabla 6.16.

Si sólo se considera el tiempo que le lleva al sistema procesar la petición del usuario, se puede observar que, al utilizar la aplicación Android, el tiempo promedio es de alrededor de 1.54 segundos para la interacción con la lámpara RGB y 2.93 segundos para la interacción con el brazo robot. La diferencia de tiempo que se observa entre ambos dispositivos se debe a limitaciones de interacción con el brazo robot, ya que no es posible realizar dos solicitudes a este dispositivo en un periodo menor a un segundo. Por esta razón, para poder llevar a cabo alguna petición que requiera el movimiento del brazo robot, se requiere realizar dos peticiones: la primera para conocer el estado actual del robot y la segunda con las nuevas coordenadas a las que el brazo robot debe moverse, generando la necesidad de esperar un segundo entre cada petición. Si esta restricción no existiera, el tiempo promedio sería de 1.93 segundos.

Como se mencionó anteriormente, en las Tablas 6.15 y 6.16, y en la Figura 6.1 se muestran tiempos relacionados con "Erro", lo que indica que ocurrió algún problema al realizar una petición, como la falta de información para llevar a cabo una acción. En este escenario, el sistema envía un mensaje al usuario, notificándole la falta de datos. Como resultado, el tiempo de respuesta es más corto. Esto se debe a que el sistema no completa todo el proceso necesario para la interacción con el dispositivo solicitado y procede sólo a enviar un mensaje de notificación al usuario, al momento de identificar el error.

6.3. Integración con el sistema OIC

El objetivo principal de integrar el sistema de reconocimiento de voz y habla con el sistema OIC es evaluar las respuestas proporcionadas por el módulo de LLM. Las respuestas generadas por este módulo están basadas en la información obtenida del vídeo del escenario de prueba que se graba y del audio adquirido mediante el sistema de captura permanente y traducido a formato de texto por el módulo de procesamiento de lenguaje natural del micro-servicio. Esta integración tiene como finalidad abordar las limitaciones de OIC mencionadas en el Capítulo 5, donde una de estas es que sólo puede recopilar información de lo que es visible en un vídeo mas no de los sonidos en el escenario (e.g., la conversación entre dos personas), lo que provoca algunas deficiencias en las respuestas obtenidas por parte del sistema OIC.

Para poder realizar este análisis, se desarrolló un escenario en el que se consideraron aspectos complejos para el sistema OIC, tales como: 1) conversación entre dos personas y 2) objetos que actualmente no son posibles de detectar por el sistema OIC.

El escenario fue grabado en un vídeo de 1.27 minutos, dado que actualmente el sistema

OIC sólo ha sido evaluado en escenarios de corta duración, menores a dos minutos, en donde dos personas se encuentran entablando una conversación sobre las actividades que uno de ellos está realizando. Dichas actividades involucran la interacción con diversos dispositivos y objetos, dentro de los cuales existe uno que no es posible de identificar por el sistema OIC (brazo robot), así como diversos detalles sobre las personas como sus nombres, preguntas, etc. En la Figura 6.2 se puede observar algunas imágenes que ejemplifican el flujo del escenario elaborado.



Figura 6.2: Ejemplificación del escenario realizado para pruebas de integración del sistema de reconocimiento de voz y habla multi-fuente con el sistema OIC

Para la evaluación del sistema, Se generaron seis preguntas, las cuales se basan en la información que puede ser adquirida mediante vídeo y audio. Las preguntas para la evaluación son las siguientes:

1. ¿Qué están haciendo los hombres?
2. ¿Qué está haciendo el hombre que está sentado en la silla?
3. ¿Por qué el hombre en la silla está usando su *smartphone*?
4. ¿Puedes decirme cuál es el contexto del escenario?
5. ¿Puedes decirme qué contiene la botella?
6. Dime el nombre de la persona a la que pertenece la botella.

Las respuestas obtenidas por OIC, proporcionando como parámetros de entrada la información obtenida con y sin el sistema de reconocimiento de voz y habla multi-fuente,

son presentadas a continuación. Las respuestas en café son las obtenidas sin la información de audio y las respuestas en turquesa son las obtenidas agregando la información de audio.

- Pregunta 1: ¿Qué están haciendo los hombres?
 1. Los hombres en la escena están sentados en sillas, vistiendo ropa y zapatos diferentes. También se les ve de pie, sosteniendo bolsas e interactuando con objetos en la habitación, como una computadora portátil, libros y una botella en un escritorio. En algunas imágenes, se les ve tomándose de las manos o usando las manos para interactuar con objetos en la escena.
 2. En la conversación, los hombres están discutiendo y mostrando una aplicación que permite la interacción por voz con dispositivos. Específicamente, hablan sobre controlar un robot, una lámpara y persianas mediante comandos de voz.
- Pregunta 2: ¿Qué está haciendo el hombre que está sentado en la silla?
 1. El hombre sentado en la silla lleva ropa y accesorios diferentes en cada imagen, por lo que es difícil determinar exactamente lo que está haciendo sin más contexto. Sin embargo, según los diferentes objetos que aparecen en algunas de las imágenes, podría estar trabajando, estudiando, leyendo o usando una computadora portátil.
 2. El hombre sentado en la silla está probando su aplicación para interactuar con dispositivos mediante comandos de voz.
- Pregunta 3: ¿Por qué el hombre en la silla está usando su *smartphone*?
 1. En la última imagen (imagen 87), el hombre sentado en la silla está usando su teléfono inteligente. Sin embargo, según la información proporcionada, no se especifica para qué está usando específicamente su teléfono inteligente. Podría ser para varios propósitos, como hacer una llamada telefónica, enviar un mensaje, revisar correos electrónicos, navegar por Internet o usar aplicaciones.
 2. El hombre en la silla está usando su teléfono inteligente para interactuar con los dispositivos a su alrededor. Está utilizando una aplicación en su teléfono para controlar y comunicarse con los dispositivos, como el robot, la lámpara, las persianas y otros dispositivos en la habitación. Puede dar comandos de voz o tocar los botones en su teléfono para realizar acciones y recibir retroalimentación de los dispositivos.
- Pregunta 4: ¿Puedes decirme cuál es el contexto del escenario?
 1. Según las descripciones proporcionadas, el contexto del escenario parece ser una serie de acciones que involucran a dos individuos, man_fc5d y man_2c4a,

en varios entornos, como una habitación, una silla y un escritorio. Se les muestra vistiendo diferentes prendas de vestir, incluyendo camisas, pantalones y zapatos. Además, hay varios objetos presentes en las escenas, como botellas, bolsas, libros, lámparas y computadoras portátiles. Las acciones involucran estar sentados, de pie, sosteniendo objetos e interactuando con el entorno.

2. El contexto del escenario es una conversación entre dos individuos, Ivan y Sanjiv, en la que discuten una aplicación que permite la interacción por voz con diversos dispositivos. Ivan le está demostrando la aplicación a Sanjiv y explicando su funcionalidad, incluyendo la capacidad de controlar un robot, una lámpara y persianas. Ivan también menciona que la aplicación proporciona retroalimentación de un micro-servicio para solucionar cualquier problema con los dispositivos.

- Pregunta 5: ¿Puedes decirme qué contiene la botella?
 1. Desafortunadamente, no puedo determinar lo que contiene la botella según la información proporcionada.
 2. La botella contiene agua.
- Pregunta 6: Dime el nombre de la persona a la que pertenece la botella.
 1. La botella pertenece a man_fc5d.
 2. La botella pertenece a Ivan.

Como se puede observar en las respuestas obtenidas, estas varían de manera considerable dependiendo de la incorporación o no de la información obtenida por el sistema de reconocimiento de voz y habla. Como ya se mencionó a principios de esta sección, cuando esta información no es proporcionada, el sistema tiende a realizar suposiciones al generar respuestas sobre algunas preguntas, tal como se puede observar en la respuesta en café a la pregunta tres, en donde se realizan suposiciones sobre la acción que realiza la persona con su *smartphone*. Sin embargo, al agregar la información de audio capturada, el sistema es capaz de identificar con mayor precisión el uso que la persona le está dando a su *smartphone* en ese momento.

Por otra parte, se ha demostrado que es posible la identificación de dispositivos u objetos que no son posibles de identificar mediante el módulo *Scene Graph* del sistema OIC. En la pregunta tres, como parte de la respuesta generada, el sistema menciona objetos que se encuentran en el escenario, como el robot y la lámpara que, sin ayuda de la información de audio, no habría podido identificar. Además, el sistema adquiere mayor entendimiento sobre el escenario y los objetos que se encuentran involucrados. La pregunta cinco hace referencia al contenido de un objeto que es capaz de ser identificado por el sistema mediante la visión, sin embargo, sin la información adicional de audio, el sistema no es capaz

de identificar su contenido. Una vez que se agrega la información adicional, el sistema es capaz de identificarlo.

En resumen, los resultados muestran que la incorporación de información de audio mejora significativamente la capacidad del sistema OIC para comprender y responder preguntas específicas basadas en la información visual y auditiva del escenario. Las respuestas con información de audio son más precisas y detalladas, lo que demuestra la utilidad de la integración del sistema de reconocimiento de voz y habla con el sistema OIC.

No obstante, una de las deficiencias que aún persiste en el sistema es la posibilidad de confusión al identificar a qué parte de la conversación transcrita pertenece cada sujeto. Esto, en ocasiones, puede dar lugar a errores, dependiendo de la pregunta formulada, ya que este podría atribuir acciones o expresiones a la persona equivocada.

Capítulo 7

Conclusiones y trabajo a futuro

Este trabajo de tesis ha sido desarrollado tomando en consideración dos de las deficiencias que suelen tener los actuales compañeros digitales, como Alexa o *Google Assistant*. La primera es la necesidad de permanecer dentro de un área delimitada por el rango de escucha de estos asistentes, siempre que se requiera de su utilización, ya que estos solamente cuentan con una fuente captadora de audio (micrófono colocado dentro de estos asistentes). Esto a su vez ocasiona que el usuario necesite adaptarse a estos asistentes para poder utilizarlos, lo que indica que esos asistentes actualmente carecen de una conciencia de contexto al no poder adaptarse a las necesidades del usuario.

La segunda deficiencia de estos asistentes se refiere a la privacidad de la información de los usuarios. Estos asistentes realizan todo el procesamiento requerido para la atención de la petición solicitada, mediante el computo en la nube, lo cual indica que toda la información de los usuarios es enviada a los servidores de las compañías que desarrollan estos asistentes, como Amazon y Google, permitiéndoles utilizar dicha información a su conveniencia.

Con base en estas dos deficiencias, se planteo la hipótesis “Es posible desarrollar un reconocedor de voz multi-fuente capaz de interactuar con los usuarios en diversos dominios de aplicación, asegurando la confidencialidad mediante el procesamiento integral de datos de forma local, sin la necesidad de compartir información con entidades externas”. Al implementar un sistema de reconocimiento de voz y habla multi-fuente, que cuenta con la capacidad de poder recibir y enviar información a diferentes sistemas y que no requiere computo en la nube, se logra abordar estas dos deficiencias anteriormente mencionadas. Por una parte, las múltiples fuentes ayudan a ampliar el rango de uso de este sistema por los usuarios. Por otra parte, la utilización de herramientas, de manera local, asegura al usuario que su información no será compartida con personas ajenas o compañías. Por esta razón, se puede afirmar que la hipótesis se valida positivamente. Sin embargo, se tiene que tener en consideración las limitantes que se pueden llegar a tener, como el uso

de dispositivos con las capacidades necesarias para la implementación del micro-servicio encargado del procesamiento de las peticiones de los usuarios, ya que tiende a ser costoso desde el punto de vista computacional, especialmente en la traducción del formato de audio a texto.

7.1. Conclusiones

Las conclusiones de este trabajo de investigación se listan a continuación:

- Las fuentes de captura de audio, utilizadas para la captura de las peticiones de los usuarios, permiten la realización de dichas peticiones en un área mayor a la de los asistentes que solamente cuentan con una sola fuente de entrada. Si bien el rango de los sistemas de captura permanente está limitado a distancias no mayores a dos o dos metros y medio, es posible la implementación de varios de estos sistemas de manera estratégica, que permitan obtener la petición del usuario, e.g., en el sector doméstico, un dispositivo por cada cuarto o área de la casa como cocina, comedor, sala, etc, permiten la atención de peticiones en todo el hogar. Aunado a esto, el uso de la aplicación Android permite realizar peticiones en áreas abiertas, como en un jardín, en donde el ruido pudiera afectar en mayor medida a los sistemas de captura permanente.
- La implementación de múltiples sistemas de captura permanente es una forma viable para el desarrollo de un sistema multi-fuente, ya que solamente se requiere un dispositivo de bajo consumo como el ESP32-S3 y el micrófono digital INMP441. Estos artefactos no generan un gran costo económico, en comparación con la adquisición de dos asistentes digitales para la simulación de estas múltiples fuentes de entrada. Al tener dos asistentes como Alexa, cada asistente procesa sus propias peticiones, en comparativa con el sistema de captura permanente en donde todas las peticiones capturadas son enviadas a un solo lugar que, en este caso, es el micro-servicio propuesto en esta tesis.
- El uso de herramientas como *Whisper* y *SpaCy* que permiten su utilización, sin la necesidad de recurrir al cómputo en la nube. Sólo se requiere descargar sus herramientas como los módulos para la traducción de audio a texto o identificación de palabras, mediante la generación de patrones. Además, el uso de estas herramientas permite que la información de usuario se mantenga en un entorno local, ya que no es compartida con ningún tercero como alguna compañía. Sin embargo, al mantener la información de manera local, se requiere un dispositivo que contenga la capacidad de memoria suficiente para almacenar dicha información, así como los recursos computacionales para poder utilizar las herramientas de manera fluida.

- El sistema de reconocimiento de voz y habla, desarrollado en esta tesis, tiene la flexibilidad para ser implementado en escenarios que no se encuentren totalmente relacionados y que requieran la captura y el procesamiento de información de audio.
- Es posible la implementación de herramientas como OpenVino que ayuden a acelerar el procesamiento realizado por los *frameworks* de procesamiento de lenguaje natural para la atención de las peticiones de usuarios, permitiendo utilizar dispositivos que no cuenten con tarjetas aceleradoras como GPUs de NVIDIA.

7.2. Trabajo futuro

Algunas ideas de trabajo futuro que permitirían hacer avanzar esta propuesta de investigación son las siguientes:

- Generación de un contenedor Docker que ayude a la implementación del micro-servicio en cualquier dispositivo, sin la necesidad de instalar nuevas dependencias, versiones de bibliotecas o algún sistema operativo.
- Implementación de una red neuronal convolucional para la identificación de la palabra de activación utilizada por el sistema de captura permanente, recurriendo a modelos ligeros para su implementación en dispositivos de bajo consumo, como el ESP32-S3. Esto mejoraría la velocidad de respuesta actual ya que, al realizar la identificación de la palabra de activación en el sistema de captura permanente, se evitaría el envío constante de peticiones al micro-servicio y, por ende, la espera de la respuesta para determinar si se ha dicho o no la palabra de activación.
- Incremento del número de sistemas de captura permanente en el sistema de reconocimiento de voz y habla para su evaluación en escenarios más amplios.
- Incremento del número de sistemas de captura permanente en el sistema OIC para la captura de información en escenarios más complejos, así como la generación de un archivo de texto que contenga la información obtenida por cada uno de los sistemas de captura permanente de manera organizada. Este archivo podrá ser integrado a la información procesada por el sistema OIC y se podrán realizar pruebas que permitan comparar los resultados obtenidos con los actuales y observar si estos mejoran.

Bibliografía

- [Fre,] Freertos real time operating system (rtos). <https://www.freertos.org/>. Accessed: 13 de junio del 2023.
- [Alagha and Helbing, 2019] Alagha, E. C. and Helbing, R. R. (2019). Evaluating the quality of voice assistants' responses to consumer health questions about vaccines: an exploratory comparison of alexa, google assistant and siri. *BMJ HEALTH & CARE INFORMATICS*, 26(1).
- [Avola, 2009] Avola, D. (2009). *Context Modelling Approaches for Mobile Systems*, pages 364–378. Handbook of Research on Mobile Multimedia, Second Edition. IGI Global, Hershey, PA, USA.
- [Boyd and Wilson, 2018] Boyd, M. and Wilson, N. (2018). Just ask siri? a pilot study comparing smartphone digital assistants and laptop google searches for smoking cessation advice. *PLOS ONE*, 13(3):1–6.
- [Brewer et al., 2022] Brewer, R., Pierce, C., Upadhyay, P., and Park, L. (2022). An empirical study of older adult's voice assistant use for health information seeking. *ACM TRANSACTIONS ON INTERACTIVE INTELLIGENT SYSTEMS*, 12(2, SI).
- [Cheng et al., 2018] Cheng, A., Raghavaraju, V., Kanugo, J., Handrianto, Y. P., and Shang, Y. (2018). Development and evaluation of a healthy coping voice interface application using the google home for elderly patients with type 2 diabetes.
- [De la Fraga, 2001] De la Fraga, L. (2001). La transformada discreta de fourier y la transformada rápida de fourier. pages 1–22.
- [Dojchinovski et al., 2019] Dojchinovski, D., Ilievski, A., and Gusev, M. (2019). Interactive home healthcare system with integrated voice assistant. In *2019 42ND INTERNATIONAL CONVENTION ON INFORMATION AND COMMUNICATION TECHNOLOGY, ELECTRONICS AND MICROELECTRONICS (MIPRO)*, pages 284–288.
- [Ermolina, 2021] Ermolina, A. (2021). Voice-controlled intelligent personal assistants in health care: International delphi study. *JOURNAL OF MEDICAL INTERNET RESEARCH*, 23(4).

- [Espressif, 2023] Espressif (2023). Esp32-s3 series. Technical report, Espressif Systems.
- [Eugenio, 2019] Eugenio, G. (2019). Introducción a las redes neuronales de convolución. aplicación a la visión por ordenador.
- [Floridi, 2018] Floridi, L. (2018). Soft ethics and the governance of the digital. *Philosophy & Technology*, 31(1):1–8.
- [Foltynski et al., 2018] Foltynski, P., Ladyzynski, P., Pankowska, E., and Mazurczak, K. (2018). Efficacy of automatic bolus calculator with automatic speech recognition in patients with type 1 diabetes: A randomized cross-over. *Journal of Diabetes*, 10(7):600–608.
- [Gamma, 2002] Gamma, E. (2002). *Patrones de diseño: elementos de software orientado a objetos reutilizable*. Addison-Wesley professional computing series. Pearson Educación.
- [Gan, 2020] Gan, W. S. (2020). *Fast Fourier Transform*, pages 17–20. Springer Singapore, Singapore.
- [Garcia et al., 2020] Garcia, K., Mayer, S., Ricci, A., and Ciortea, A. (2020). Proactive digital companions in pervasive hypermedia environments. In *2020 IEEE 6TH INTERNATIONAL CONFERENCE ON COLLABORATION AND INTERNET COMPUTING (CIC 2020)*, pages 54–59, 345 E 47TH ST, NEW YORK, NY 10017 USA. IEEE; IEEE Comp Soc, IEEE.
- [González, 2018] González, M. (2018). Desarrollo de un sistema de cancelación acústico para mejorar la ergonomía en el trabajo en entornos industriales.
- [Goodfellow et al., 2016] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. <http://www.deeplearningbook.org>.
- [Gourley et al., 2002] Gourley, D., Totty, B., Sayer, M., Aggarwal, A., and Reddy, S. (2002). *HTTP: The Definitive Guide: The Definitive Guide*. Definitive Guides. O’Reilly Media.
- [Habibi Aghdam and Jahani Heravi, 2017] Habibi Aghdam, H. and Jahani Heravi, E. (2017). *Convolutional Neural Networks*, pages 85–130. Springer International Publishing, Cham.
- [Hong et al., 2009] Hong, J.-y., Suh, E.-h., and Kim, S.-J. (2009). Context-aware systems: A literature review and classification. *EXPERT SYSTEMS WITH APPLICATIONS*, 36(4):8509–8522.
- [Honnibal et al., 2020a] Honnibal, M., Montani, I., Landeghem, S. V., and Boyd, A. (2020a). spacy: Industrial-strength natural language processing in python. *Zenodo*.

- [Honnibal et al., 2020b] Honnibal, M., Montani, I., Landeghem, S. V., and Boyd, A. (2020b). *spacy: Industrial-strength natural language processing in python*. *Zenodo*.
- [IBM, 2020] IBM (2020). <https://www.ibm.com/cloud/learn/speech-recognition>. Accessed: 2022-11-01.
- [INEGI, 2021] INEGI (2021). Estadísticas a propósito del día mundial de la diabetes (14 de noviembre) datos nacionales.
- [integrated, 2014] integrated, M. (2014). Microphone amplifier with agc and low-noise microphone bias. Technical report, Adafruit Industries.
- [InvenSense, 2014] InvenSense (2014). Omnidirectional microphone with bottom port and i2s digital output. Technical report, TDK InvenSense.
- [Jesus-Azabal et al., 2020] Jesus-Azabal, M., Rojo, J., Moguel, E., Flores-Martin, D., Berrocal, J., Garcia-Alonso, J., and Murillo, J. M. (2020). Voice assistant to remind pharmacologic treatment in elders. In *GERONTECHNOLOGY, IWOG 2019*, volume 1185 of *Communications in Computer and Information Science*, pages 123–133.
- [Julius, 2014] Julius (2014). http://julius.osdn.jp/en_index.php. Accessed: 2022-10-19.
- [Karimi et al., 2022] Karimi, P., Ballard, K., Vazirani, P., Jorigay, R. T. N., and Martin-Hammond, A. (2022). Designing conversational assistants to support older adults’ personal health record access. pages 253–271. SPRINGER INTERNATIONAL PUBLISHING AG. 15th European-Alliance-for-Innovation (EAI) International Conference on Pervasive Computing Technologies for Healthcare (Pervasive Health), ELECTRONETWORK.
- [Kritzler et al., 2019] Kritzler, M., Hodges, J., Yu, D., Garcia, K., Shukla, H., and Michahelles, F. (2019). Digital companion for industry artificial meets human intelligence. In *COMPANION OF THE WORLD WIDE WEB CONFERENCE (WWW 2019)*, pages 663–667.
- [Larman, 2005] Larman, C. (2005). *Applying UML and patterns-An introduction to Object-Oriented Analysis and Design and Iterative Development*, volume 3er edition. Prentice Hall.
- [Latif et al., 2021] Latif, S., Qadir, J., Qayyum, A., Usama, M., and Younis, S. (2021). Speech technology for healthcare: Opportunities, challenges, and state of the art. *IEEE REVIEWS IN BIOMEDICAL ENGINEERING*, 14:342–356.

- [León, 2019] León, J. (2019). *Redes informáticas: Protocolos de comunicación, protocolo de Aplicación y Software: Cómo desarrollar aplicaciones para conectarse a Mainframe y servidores*. Tecnología por dentro. Jacinto Fung León.
- [Martinelli, 2022] Martinelli, J. E. (2022). Clasificación de datos desbalanceados.
- [Mensah and Mwakapesa, 2022] Mensah, I. K. and Mwakapesa, D. S. (2022). The impact of context awareness and ubiquity on mobile government service adoption. *Mobile Information Systems*, 2022:5918826.
- [Miner et al., 2016] Miner, A. S., Milstein, A., Schueller, S., Hegde, R., Mangurian, C., and Linos, E. (2016). Smartphone-Based Conversational Agents and Responses to Questions About Mental Health, Interpersonal Violence, and Physical Health. *JAMA Internal Medicine*, 176(5):619–625.
- [Munier et al., 2014] Munier, M., Lalanne, V., Ardoy, P.-Y., and Ricarde, M. (2014). Legal issues about metadata data privacy vs information security. In Garcia-Alfaro, J., Lioudakis, G., Cuppens-Boulahia, N., Foley, S., and Fitzgerald, W. M., editors, *Data Privacy Management and Autonomous Spontaneous Security*, pages 162–177, Berlin, Heidelberg. Springer Berlin Heidelberg.
- [National Institute of Standards and Technology, 2004] National Institute of Standards and Technology (2004). Standards for security categorization of federal information and information systems. Technical report, U.S. Department of Commerce, Washington, D.C.
- [OpenAI, 2022] OpenAI (2022). Whisper.
- [OPS, 2021] OPS (2021). Diabetes. <https://www.paho.org/es/temas/diabetes>. Accessed: 2022-10-12.
- [Orfanidis, 2010] Orfanidis, S. J. (2010). *Introduction to Signal Processing*. Prentice Hall.
- [Palermos, 2023] Palermos, S. O. (2023). Data, metadata, mental data? privacy and the extended mind. *AJOB Neuroscience*, 14(2):84–96.
- [Palma et al., 2003] Palma, J., Garrido, M. d. C., Sánchez, F., and Quesada, A. (2003). *Programación Concurrente*. Thomson.
- [Proakis, 2006] Proakis, J. G. (2006). *Digital Signal Processing: Principles, Algorithms, and Applications*. Prentice Hall.
- [Pérez, 2011] Pérez, R. (2011). Sistema caracterizador de equipos de audio (scea).

- [Radford et al., 2023] Radford, A., Kim, J. W., Xu, T., Brockman, G., Mcleavey, C., and Sutskever, I. (2023). Robust speech recognition via large-scale weak supervision. In Krause, A., Brunskill, E., Cho, K., Engelhardt, B., Sabato, S., and Scarlett, J., editors, *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 28492–28518. PMLR.
- [Rehman et al., 2020] Rehman, U. U., Chang, D. J., Jung, Y., Akhtar, U., Razzaq, M. A., and Lee, S. (2020). Medical instructed real-time assistant for patient with glaucoma and diabetic conditions. *Applied Sciences*, 10(7).
- [Renjith and Manju, 2017] Renjith, S. and Manju, K. G. (2017). Speech based emotion recognition in tamil and telugu using lpcc and hurst parameters. In *PROCEEDINGS OF 2017 IEEE INTERNATIONAL CONFERENCE ON CIRCUIT ,POWER AND COMPUTING TECHNOLOGIES (ICCPCT)*.
- [Roca et al., 2021] Roca, S., Lozano, M. L., Garcia, J., and Alesanco, A. (2021). Validation of a virtual assistant for improving medication adherence in patients with comorbid type 2 diabetes mellitus and depressive disorder. *INTERNATIONAL JOURNAL OF ENVIRONMENTAL RESEARCH AND PUBLIC HEALTH*, 18(22).
- [Salai et al., 2021] Salai, A.-M., Cook, G., and Holmquist, L. E. (2021). Intravox: A personalized human voice to support users with complex needs in smart homes. In *HUMAN-COMPUTER INTERACTION, INTERACT 2021, PT I*, volume 12932, pages 223–244.
- [Seco, 2019] Seco, G. (2019). *La transformada discreta de Fourier*. : Oberta UOC Publishing, SL, Barcelona.
- [Smith, 1999] Smith, S. W. (1999). *The Scientist and Engineer’s Guide to Digital Signal Processing*. California Technical Publishing.
- [Torres, 2020] Torres, J. (2020). *Python Deep Learning: Introducción práctica con Keras y TensorFlow 2*. Alpha Editorial.
- [Weiss, 1999] Weiss, G. (1999). *Multiagent systems: a modern approach to distributed artificial intelligence*. The MIT Press.
- [Whitman and Mattord, 2021] Whitman, M. E. and Mattord, H. J. (2021). *Principles of Information Security*. Cengage learning, 6 edition.
- [Wilson et al., 2017] Wilson, N., MacDonald, E. J., Mansoor, O. D., and Morgan, J. (2017). In bed with siri and google assistant: a comparison of sexual health advice. *BMJ*, 359.

[WoT Working Group, 2023] WoT Working Group (2023). Web of things (wot) thing description. Technical report, World Wide Web Consortium (W3C).