

TRABAJO DE FINAL DE GRADO

Grado en Ingeniería Electrónica Industrial y Automática (EEIA)

CHATBOT PARA ENFERMEDADES MINORITARIAS



Memoria y Anexos

Autor/a:	Víctor Martínez Amador
Director/a:	Gerard Escudero Bakx
Co-Director/a:	Samir Kanaan Izquierdo
Convocatoria:	Mayo 2021

Resumen

El presente proyecto consiste en la confección de un chatbot diseñado para dar soporte a pacientes y familiares afectados por enfermedades raras, con la finalidad de cubrir una necesidad tan fundamental como la obtención de información de calidad, veraz, y contrastada. Partiendo de un prototipo ya existente creado como punto de partida para el proyecto europeo *Share4Rare* contra las enfermedades raras, se analiza la propuesta inicial con el objetivo de alcanzar una versión mejorada que consolide una base más sólida para el chatbot, focalizando en aspectos importantes como la información veraz, la estructuración y gestión de los datos y el procesamiento del lenguaje.

El proyecto se ha enfocado con una metodología de trabajo basada tanto en investigación cualitativa como cuantitativa, esto es, se ha ido analizando las cualidades del bot y los recursos disponibles para crear una versión mejorada y, al mismo tiempo, se han utilizado datos recolectados para justificar estos cambios.

El resultado ha sido un bot con información de enfermedades raras ampliada, capaz de gestionar la información desde una base de datos con capacidad de conexión con un servidor en línea, sin depender de archivos locales, que además presenta un sistema de procesamiento del lenguaje más preciso que ofrece una mayor tasa de acierto frente a las consultas de los usuarios.



Resum

El present projecte consisteix en la confecció d'un chatbot dissenyat per donar suport a pacients i familiars afectats per malalties rares, amb la finalitat de cobrir una necessitat tan fonamental com l'obtenció d'informació de qualitat, veraç, i contrastada. Partint d'un prototip ja existent creat com a punt de partida per al projecte europeu Share4Rare contra les malalties rares, s'analitza la proposta inicial amb l'objectiu d'assolir una versió millorada que consolidi una base més sòlida per al chatbot, focalitzant en aspectes importants com la informació veraç, l'estructuració i gestió de les dades i el processament del llenguatge.

El projecte s'ha enfocat amb una metodologia de treball basada tant en investigació qualitativa com quantitativa, és a dir, s'ha anat analitzant les qualitats del bot i els recursos disponibles per crear una versió millorada i, al mateix temps, s'han utilitzat dades recollides per justificar aquests canvis.

El resultat ha estat un bot amb informació de malalties rares ampliada, capaç de gestionar la informació des d'una base de dades amb capacitat de connexió amb un servidor en línia, sense dependre d'arxius locals, que a més presenta un sistema de processament del llenguatge més precís que ofereix una major taxa d'encert davant de les consultes dels usuaris.

Abstract

This project consists of the development of a chatbot designed to provide support to patients and relatives affected by rare diseases, with the aim of covering such a fundamental need as obtaining quality, truthful and contrasted information. Based on an existing prototype created as a starting point for the European project Share4Rare against rare diseases, the initial proposal is analysed with the objective of achieving an improved version that consolidates a more solid foundation for the chatbot, focusing on important aspects such as accurate information, data structuring and management and language processing.

The project has been approached with a working methodology based on both qualitative and quantitative research, i.e., the qualities of the bot and the available resources have been analysed to create an improved version and, at the same time, collected data have been used to justify these changes.

The result has been a bot with expanded rare disease information, capable of managing information from a database with the capability to connect to an online server, without relying on local files, which also has a more accurate language processing system that offers a higher hit rate against user queries.



Agradecimientos

Primeramente, agradecer a Gerard Escudero Bakx por sugerirme el tema del proyecto cuando acudí a él para asesorarme sobre el TFG. Agradecerle también, tanto a él como a Samir Kanaan Izquierdo, los tutores de este trabajo de final de grado, toda la ayuda brindada durante el transcurso del proyecto, sin ellos no podría haber logrado los resultados obtenidos.

Glosario

Hackathon: encuentro de programadores y otras personas involucradas en el desarrollo de software con el fin de diseñar y desarrollar software de manera colaborativa.

Enfermedad rara: enfermedad que afecta a menos de 5 por cada 10000 habitantes. Existen más de 6000 a día de hoy.

Melanoma pediátrico: melanoma que afecta a pacientes menores de 21 años. Supone solo el 1-4% de todos los melanomas; extremadamente raro.

ELA: Esclerosis Lateral Amiotrófica, en inglés *amyotrophic lateral sclerosis (ALS)*, es una enfermedad neurodegenerativa progresiva que afecta a las células nerviosas del cerebro y la médula espinal. Afecta aproximadamente a 1-2 personas por cada 100000 habitantes.

Geocodificación: es la conversión de una localización (coordenadas, dirección, nombre, etc) en una ubicación sobre la superficie del planeta.

Clase: en programación, es una entidad que define ciertos elementos que se caracterizan por tener unos atributos (información) y un comportamiento (operaciones).

Objeto: en programación, es una instancia de una clase (pertenece y tiene las características de esa clase).

Flag: variable utilizada en programación para almacenar un valor binario que se utiliza para establecer el estado de alguna cosa concreta.

Chatbot: es un programa con la capacidad de simular una conversación con un usuario, dar información o ejecutar ciertas acciones.

API: *Application Programming Interfaces*, en español Interfaz de Programación de Aplicaciones, es una herramienta que permite la interconexión de dos aplicaciones software, por medio de una serie de pautas.

Comando: se refiere a un tipo de interacción con los *chatbots*; son palabras concretas que deben empezar con el símbolo '/' (p. ej. /start), que sirven para ejecutar ciertas rutinas o funciones del programa.

Markdown: es un método de escritura que utiliza un tipo de sintaxis concreto para estructurar y establecer un formato y diseño al texto (títulos, negritas, cursivas, numeración, enlaces, etc).

IA: Inteligencia Artificial.

Machine learning: disciplina de la Inteligencia Artificial que utiliza algoritmos de aprendizaje autónomo para la predicción y la gestión de datos.

Deep Learning: técnicas de *Machine Learning* que utilizan estructuras parecidas a las neuronas humanas (neuronas artificiales) y que se basan en la aplicación de múltiples capas.

Correlación: valor numérico que indica la relación entre dos variables; se basa en la evaluación de la tendencia en los datos. Dos variables se correlacionan cuando tienen una tendencia creciente o decreciente entre ellas.

Redes neuronales: técnica de *Deep Learning* inspirada en el cerebro humano, que utiliza una agrupación de nodos (neuronas artificiales) que transmiten información entre ellas.

NLP: *Neuro-Linguistic Programming*, en español procesamiento del lenguaje natural, es un campo de la inteligencia artificial que estudia la interpretación y el entendimiento del lenguaje humano.

STS: *Semantic Textual Similarity*; es la similitud semántica de dos textos. Indica qué tan parecidos son dos textos entre sí.

Word Embedding: es un tipo de técnica de NLP que representa las palabras como vectores de números reales (vectorización).

Sentence Embedding: es un tipo de técnica de NLP que utiliza la vectorización de las frases como representación de estas.

Spacy: es una API de procesamiento natural del lenguaje que contiene un modelo *Word Embedding* para el cálculo de la STS.

BERT: *Bidirectional Encoder Representations from Transformers*. Algoritmo de NLP desarrollado por Google que utiliza redes neuronales con una funcionalidad *Sentence Embedding*.

Keywords: palabra muy común en el campo del NLP. Son las palabras clave de un texto, que se suelen extraer para eliminar palabras vacías (sin significado).

nltk: *Natural Language Toolkit*. Módulo de NLP muy común en la programación con Python.

Rake: Algoritmo de extracción rápida de *keywords* que utiliza *nltk*.

Database: significa base de datos, y es típicamente utilizado al hablar de ellas.

JSON: formato para el almacenamiento de datos que facilita la lectura y escritura por parte de una persona a la vez que es sencillo para las máquinas interpretarlo y generarlo.

Pipeline: proceso utilizado para ahorrar tiempo de computación; se basa en leer una instrucción mientras se está interpretando y ejecutando otras dos, en vez de esperar a que termine el proceso completo de una para empezar con la siguiente.

Pickle: módulo que permite la serialización de objetos de Python. Se puede considerar que se está empaquetando los objetos.



Serialización: proceso en el cual se transforma un objeto de un programa en una secuencia de bytes.

IQR: rango intercuartílico. Es una estimación estadística de la dispersión de un conjunto de datos.

RE: *Regular Expression*. Las expresiones regulares son patrones de coincidencia de texto formados con una sintaxis concreta.



Índice

RESUMEN	I
RESUM	II
ABSTRACT	III
AGRADECIMIENTOS	IV
GLOSARIO	V
1. PREFACIO	1
1.1. Origen del trabajo	1
1.2. Motivación	2
1.3. Requerimientos previos.....	2
2. INTRODUCCIÓN	3
2.1. Prototipo del chatbot.....	3
2.1.1. Análisis	5
2.2. Objetivos del trabajo.....	6
2.3. Alcance del trabajo	7
3. DESARROLLO DEL PROYECTO	8
3.1. Versión 0	8
3.1.1. Puesta a punto del bot	9
3.1.2. Plantilla de evaluación.....	14
3.2. Versión 1	15
3.2.1. Cambio lectura de datos	16
3.2.1.1. Propuesta 1	17
3.2.1.2. Propuesta 2	20
3.2.1.3. Análisis final.....	22
3.2.2. Mejora del sistema de búsqueda.....	24
3.2.2.1. SpaCy	25
3.2.2.2. WebBertSimilarity y ClinicalBertSimilarity.....	27
3.2.2.3. Sistema de respuesta múltiple.....	28
3.2.2.4. Características adicionales	30
3.2.2.5. Análisis final.....	32
3.2.3. Ampliación de enfermedades	35
3.2.3.1. Esclerosis Lateral Amiotrófica (ELA).....	36

3.2.3.2. Modificaciones necesarias	37
4. CONCLUSIONES	39
4.1. Recomendaciones.....	40
5. ANÁLISIS DEL IMPACTO AMBIENTAL	41
6. ANÁLISIS ECONÓMICO	42
6.1. Coste de los recursos humanos.....	42
6.2. Coste de los recursos materiales	42
6.3. Coste total asociado al proyecto	43
BIBLIOGRAFÍA	45
ANEXO A: CONTENIDO DEL ARCHIVO ADJUNTO	49
A1. Carpeta “Hospitales”.....	49
A2. Carpeta “ProcessedData”	49
A3. Carpeta “Test STS”	49
A4. Carpeta “Códigos”	50
A4.1. Archivo “bot.py”	50
A4.2. Archivo “utils.py”	50
A4.3. Archivo “preprocessing.py”	51
A4.4. Archivo “CoordinatesFinder.py”	51
A4.5. Archivo “translate.py”	52
ANEXO B: PLANTILLA DE EVALUACIÓN	53
B1. Plantilla inicial (general + melanoma)	54
B2. Líneas de conversación adicionales (ELA + múltiple respuesta)	62

1. Prefacio

1.1. Origen del trabajo

Share4Rare, proyecto financiado por una subvención de la Comisión Europea, nació con el fin de crear una comunidad que integre a todo tipo de gente involucrada en las enfermedades raras, desde pacientes y sus familiares hasta investigadores de las mismas. Su objetivo es el de crear impacto en las enfermedades raras, mediante la divulgación de conocimiento e información veraz y el impulso de su investigación, buscando así una mejora en la calidad de vida de las personas afectadas por estas.

Con la Fundación Sant Joan de Déu a la cabeza del proyecto, Òmada Interactiva en colaboración con la Universitat Politècnica de Catalunya (UPC), ha diseñado y desarrollado la plataforma Share4Rare, con el fin de gestionar grandes cantidades de datos sobre los usuarios, siendo esto uno de los paquetes de trabajo en los que se divide el proyecto. Dentro de este paquete, se programaron dos *hackathones* con el objetivo de implicar a la comunidad en el desarrollo de herramientas que serían incluidas en la plataforma; el primero de estos tuvo lugar en Julio de 2019, bajo el nombre de *RareHacks*, que fue organizado por la UPC, con la finalidad de crear un *chatbot* que solventara los problemas de acceso a información fiable y contrastada de las enfermedades raras. La prueba, que se focalizó en el melanoma pediátrico, acabó con la presentación de 6 proyectos de *chatbot* por parte de diversos equipos.

Para continuar con el desarrollo de la herramienta que se creó en el *hackathon*, ya que como objetivo del proyecto Share4Rare se pretende incluirla en la plataforma, los tutores del presente proyecto, Gerard Escudero Bakx y Samir Kanaan Izquierdo (uno de los jueces del *RareHacks*), propusieron a un estudiante de la Escola Tècnica Superior d'Enginyeria Industrial de Barcelona (ETSEIB), Marçal Tallada Castañé, que continuase con el desarrollo de un *chatbot* con las funcionalidades propuestas en el *hackathon* como su trabajo final de carrera. Así, Marçal creó un chatbot, basado en la propuesta de los ganadores del *RareHacks*, que funcionaba a modo de prototipo para futuras versiones del mismo (1).

El presente proyecto parte de todo lo explicado en esta sección; de igual manera que pasó con Marçal, a causa de la idea aportada por parte de los tutores del proyecto, se decidió continuar con el desarrollo del *chatbot*, partiendo del prototipo ya creado por Marçal, con la finalidad de acercar este *chatbot* un poco más a una herramienta real que se pueda incluir en la plataforma del proyecto europeo Share4Rare.

1.2. Motivación

Entre el 3,5% y el 5,9% de la población mundial se ve afectada por enfermedades raras (2). El difícil diagnóstico de estas hace que los pacientes que las sufren pasen por situaciones difíciles a lo largo de sus vidas, desde problemas físicos causados directamente por las enfermedades hasta problemas psicológicos y sociales. El proyecto Share4Rare nació para solventar estos problemas, conectando personas del todo el mundo e incentivando la investigación, con el fin de mejorar la vida de las personas. El presente trabajo se ve impulsado por la misma motivación; se busca aportar un poco más al desarrollo de un *chatbot* que funcione como una herramienta que se pueda implementar en la plataforma Share4Rare y que sea de utilidad para visibilizar las enfermedades raras y dar apoyo tanto a pacientes y familiares de estas.

1.3. Requerimientos previos

Dada la naturaleza y el trasfondo de este trabajo, es necesario entender con claridad el origen del proyecto Share4Rare (3) y los eventos transcurridos en el *hackathon* organizado para crear un *chatbot* (4). Es por tanto requisito indispensable consultar el proyecto realizado por Marçal Tallada (1) que creó una base para la construcción de una versión funcional de la herramienta, siendo este el punto de partida del presente proyecto, tal como fue el *RareHacks* el punto de partida de Marçal. A pesar de esto, en la Introducción que se realiza posteriormente a este apartado, se explica y analiza el prototipo del cual se parte para establecer con mayor claridad los objetivos y el alcance que presentará el proyecto.

2. Introducción

Para poder establecer correctamente los objetivos, requisitos y alcance del proyecto es necesario antes prestar especial atención al prototipo del chatbot creado por Marçal Tallada, que es el punto de partida del mismo. De esta manera, realizando un análisis de los puntos fuertes y débiles de esta primera versión prematura de la herramienta, se podrá evaluar las necesidades más prioritarias y actuar en consecuencia a la hora de progresar en el proyecto.

2.1. Prototipo del chatbot

Primeramente, hay que aclarar en qué basó Marçal la realización de su prototipo; en la sección 4 de la memoria que redactó (1) hizo un análisis de las diferentes propuestas de *chatbot* que se presentaron en el *RareHacks* por parte de los diversos equipos que participaron. A partir de ahí, decidió utilizar la propuesta de los ganadores del *hackathon* como base de su proyecto, que ya incorporaba un sistema para proveer de información sobre el melanoma, la utilización de la geolocalización del usuario para informar del centro médico más cercano en el que tratan la enfermedad y la capacidad multilenguaje gracias al módulo traductor de Google, utilizando de información interna textos en inglés. Este *chatbot* se diseñó para funcionar en Telegram gracias a la *API* existente de este servicio de mensajería, utilizando el lenguaje de programación Python.

Marçal propuso una gran cantidad de objetivos en la sección 5 de su memoria (1), aunque algunos de ellos eran más enfocados a largo plazo; los objetivos principales que se cumplieron fueron el facilitar el acceso a información contrastada de las enfermedades raras y ayudar a encontrar con facilidad los centros médicos donde tratan las enfermedades, que son características ya presentes en el *chatbot* que presentó el equipo ganador del *RareHacks*. Como objetivos más técnicos, se centró en la facilidad de interacción con el *bot*, en la escalabilidad de los datos de enfermedades y de funcionalidades y en la capacidad multilenguaje para permitir que la herramienta pueda tener un alcance de personas mayor. En definitiva, su proyecto se centró en pulir la propuesta de *chatbot* con la que empezó a trabajar, mejorando ciertos aspectos y dejándola preparada para la adición de nuevas características.

El resultado que consiguió se plasma en la sección 7 (1) y es el prototipo del cual parte este proyecto. El primer aspecto a analizar es la arquitectura, esto es, la estructura y lógica interna del *chatbot*, incluyendo las líneas de conversación que es capaz de seguir. La primera vez que se inicia la conversación (se ejecuta el comando “/start”), el *bot* da la bienvenida y pide al usuario que escriba un texto en su idioma, con la finalidad de identificarlo, ya que puede ser que la información de Telegram sobre la región de ese usuario no sea correcta o quizás se desee hablar en un idioma distinto. Una vez hecho esto, el *chatbot* da la bienvenida oficial y solicita que se indique el nombre de la enfermedad

rara de la cual se quiere información. A partir de este punto se sigue una estructura de árbol de decisiones que se rige por el tipo de mensaje que se envíe, pudiendo ser un texto, un comando o una ubicación geográfica; el resto de tipos de mensaje (fotografías, videos, emoticonos, etc.) no están previstos en este prototipo y, por tanto, si se utilizan, el *bot* no reaccionará ante ellos y no realizará ninguna acción.

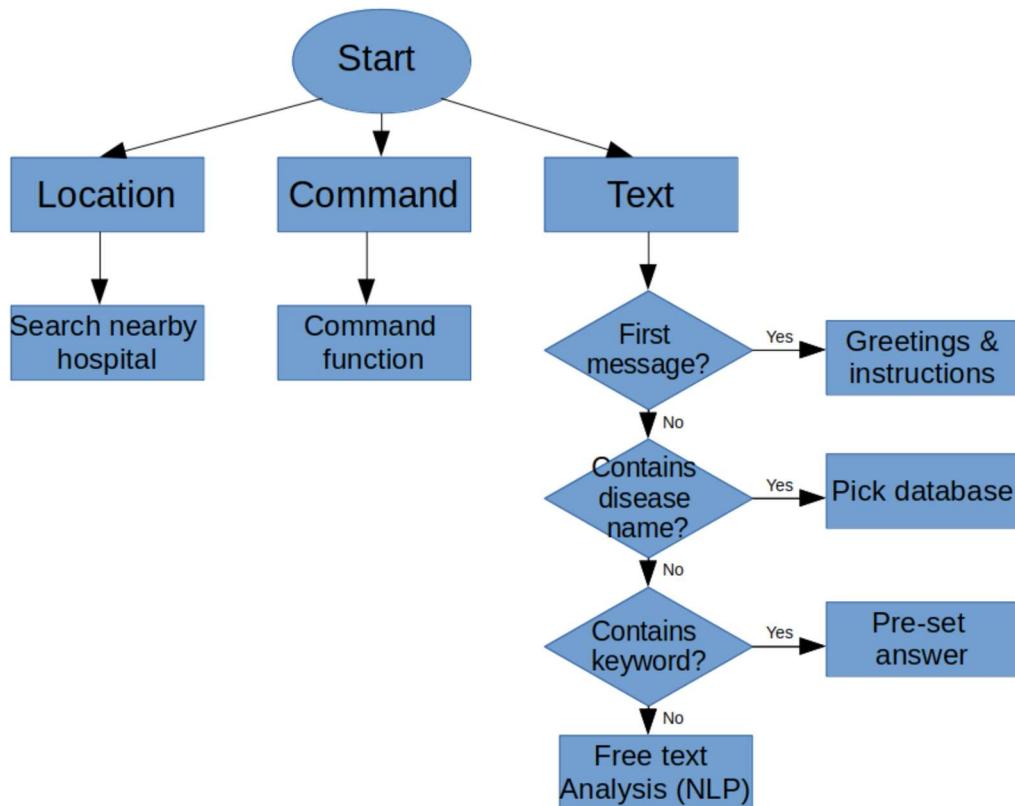


Figura 1. Diagrama de flujo de las posibles líneas de conversación con el prototipo del chatbot (1).

El texto enviado es procesado para devolver una respuesta adecuada a la consulta del usuario (sección 7.3 (1)); primero se comprueba si en el texto aparecen una serie de *keywords* concretas, que se utilizan para enviar unos mensajes preestablecidos en función de estas. En total, con este conjunto de mensajes, puede contestar a tres tipos de interacciones: un saludo (si se escribe '*hi*', '*hello*' o '*greetings*'), un agradecimiento ('*thank*', '*thanks*' o '*appreciate*') y una despedida ('*bye*' o '*goodbye*'). Si se escribe cualquier otro texto, se procesa el mensaje para compararlo con la base de datos de la enfermedad y encontrar la entrada de información de mayor coincidencia, esto es, se lleva a cabo un análisis con técnicas de NLP para realizar el cálculo de la similitud semántica (STS) y devolver la información que mejor resuelva las dudas del usuario; para esto el prototipo tiene integrado Spacy, un módulo de procesamiento del lenguaje natural que utiliza un método *Word Embedding*.

Si se envía la ubicación a través del chat, el *bot* realiza un procesamiento de la localización (sección 7.4 (1)), con el fin de encontrar el hospital más cercano en el que se trata la enfermedad de la cual se ha pedido información. Una vez encontrado el hospital, se devuelve un mensaje con la ubicación de este, junto a un mensaje que aporta información adicional.

En la sección 7.5 (1) se definen los comandos específicos que sirven para ejecutar ciertas acciones por parte del *bot*. El primero de ellos es el que ya se ha comentado (“/start”), que sirve para iniciar la conversación. Existen otros dos comandos, “/disease” y “/AllHospitals”, que se pueden utilizar para cambiar la enfermedad sobre la que se está hablando y para recibir una lista con todos los hospitales disponibles, respectivamente.

Toda la información que utiliza el *chatbot* se recoge de archivos JSON, obtenidos previamente gracias a un preprocesamiento de los datos, tanto de la enfermedad como de los hospitales. El formato de los datos y el procesamiento de algunos de ellos se explican en la sección 7.6 (1).

2.1.1. Análisis

Después de haber visto todas las características del prototipo de *chatbot* creado por Marçal, es necesario analizar cuáles son los puntos fuertes y débiles de este, para poder establecer de forma más clara los objetivo y el alcance de este proyecto. Se resaltarán los aspectos más destacables a primera vista, pero, por supuesto, el margen de mejora del *chatbot* es aún muy elevado.

Se ha visto que el *bot* utiliza una serie de archivos JSON para extraer la información de las enfermedades y de los hospitales y, a pesar de que es un formato muy sencillo y cómodo de utilizar, no es ni el más eficiente ni lo más adecuado al depender de un archivo almacenado localmente en un ordenador. Sin embargo, la estructura de los datos permite añadir o modificar con facilidad la información que se desea poner al alcance del *chatbot*.

Por otro lado, el sistema de selección de respuestas utiliza Spacy como herramienta para aplicar técnicas de NLP para el procesamiento del texto. Aunque esta herramienta es de muy fácil aplicación y provee unos resultados aceptables, utiliza un método *Word Embedding*, que no tiene en cuenta el contexto del texto, y tratándose de información médica es importante que la precisión a la hora de devolver una respuesta sea la mayor posible. Lo positivo de esta característica es que ha permitido crear toda la estructura del programa de forma sencilla, facilitando una futura mejora.

Por último, la única enfermedad de la cual el *chatbot* puede informar es el melanoma, y sería necesario ampliar el número de enfermedades para asegurar que la herramienta está preparada para gestionar distintas enfermedades al mismo tiempo; el prototipo de Marçal está diseñado para facilitar esto, por lo que, de nuevo, la escalabilidad es un punto muy destacado en su diseño.

2.2. Objetivos del trabajo

Una vez analizado el prototipo del cual parte este proyecto, se puede proceder a establecer los objetivos que se tendrán presentes a lo largo del transcurso del mismo.

Primeramente, ya que se va a partir de un programa ya existente, será necesario realizar una revisión del funcionamiento de este, para asegurar que el comportamiento y las características que se describen en la memoria de Marçal sean correctas. Por tanto, el primer objetivo que se establece es la revisión y puesta a punto del *chatbot*.

Si esta herramienta pretende ser implementada en la plataforma Share4Rare, es necesario que la información a la cual el *bot* tiene acceso se almacene en una base de datos con capacidad de almacenaje en la nube, ya que esto permitiría un fácil acceso desde cualquier parte, pudiendo así subirse el programa a un servidor. El segundo de los objetivos será implementar un sistema de gestión de datos que mantenga la información almacenada en una base de datos y que permita una fácil extracción de esta cuando el *bot* lo necesite.

Es importante que la precisión a la hora de responder a las consultas de los usuarios sea lo más elevada posible, ya que se busca conseguir una herramienta que ayude de manera eficaz a posibles pacientes o familiares de estos que se vean afectados por las enfermedades raras. Se establece como objetivo implementar sistemas de procesamiento del lenguaje que mejoren la experiencia del usuario.

Si se pretende que la herramienta sea implementada de forma consistente, será necesario informar de tantas enfermedades como sea posible, consiguiendo así ayudar a la mayor cantidad de gente posible. Es por eso que un objetivo más será el incremento de enfermedades disponibles para informar por parte del *chatbot*.

Así, a forma de resumen, los objetivos establecidos para el presente proyecto son los siguientes:

- La puesta a punto del prototipo del *chatbot*.
- La implementación de una base de datos.
- La mejora del sistema de selección de respuestas.
- Añadir información veraz de nuevas enfermedades.

2.3. Alcance del trabajo

La creación de un *chatbot* que sirva de herramienta para el proyecto europeo Share4Rare es una labor que conlleva un volumen de trabajo considerable, sobre todo si se busca lograr un comportamiento pulido y una gran cantidad de características y funcionalidades. Por esa razón, este proyecto pretende ser un paso más para lograr que la herramienta alcance un estado óptimo para ser integrado en la plataforma Share4Rare, ayudando así a todas esas personas que se ven afectadas por las poco comunes enfermedades raras.

El alcance del presente proyecto se basa en los objetivos establecidos anteriormente; se pretende mejorar el *chatbot* un poco más, focalizando en la mejora de las funciones actuales y encaminando la estructura del mismo para que, en un futuro, sea más sencillo alcanzar una versión completa y eficiente de esta herramienta.

A parte de buscar la implementación de todas las características establecidas en los objetivos, se gestionarán todos los problemas que se encuentren en el transcurso del proyecto, ya que, al estar modificando un programa ya existente, cabe la posibilidad de que aparezcan problemas y errores que pasaron desapercibidos en la creación del prototipo. Así mismo, una vez se haya terminado el desarrollo del proyecto, se pretende remarcar posibles errores que se hayan detectado y mejoras e implementaciones que se vean interesantes de cara a un futuro.



3. Desarrollo del proyecto

Por la naturaleza del proyecto planteado, se debe desarrollar de una manera pautada y estructurada, ya que puede haber múltiples alternativas para abordarlo y sino podría ser caótico. Es por ello que, para seguir un desarrollo claro y correcto, se ha decidido separar los avances y mejoras que se vayan a hacer en el proyecto en dos bloques, considerando que, después de cada bloque completado, se habrá logrado desarrollar una nueva versión del bot. Cada bloque será constituido con el fin de alcanzar los objetivos establecidos en el apartado [2.2](#).

La primera versión se centrará en la puesta a punto del *bot*, y se considerará que se habrá obtenido la versión 0, nombrada así por no tener ninguna mejora ni cambio respecto al original, siendo solo una revisión del comportamiento y las funcionalidades. El segundo bloque presentará el contenido más importante, como son las mejoras y ampliaciones propuestas en los objetivos; se considerará como la versión 1, que marcará un precedente para una futura versión mejorada, de ahí que se comience una numeración.

Dado que se ha partido del prototipo analizado anteriormente, se irán haciendo referencias continuas a la memoria de Marçal para destacar puntos concretos relacionados con las modificaciones que se irán realizando. También, como cabría esperar, junto a este documento se entrega un archivo adjunto ([Anexo A](#)) que contiene todos los archivos utilizados en el transcurso del proyecto, incluyendo el programa necesario para ejecutar el *chatbot*; esto también será referenciado en varias ocasiones para indicar el lugar en el que han sucedido los cambios o las mejoras.

3.1. Versión 0

Antes de empezar con los cambios realmente significativos del programa, y dado el origen del proyecto, se deberá hacer ciertas comprobaciones para asegurar el correcto funcionamiento del prototipo sobre el cual se va a partir. Para ello simplemente se procederá a ejecutar el programa y a entablar una conversación con el *bot* para ver si todo funciona bien. La resolución de todos los posibles errores que se detecten será el primer paso que dar en el transcurso del proyecto, y una vez completado se considerará que se ha alcanzado la versión 0 del bot.

Una vez acabada esta versión, pero dentro de este mismo bloque, se procederá a hacer una plantilla de evaluación, que consistirá en seguir y registrar ciertos hilos de conversación con el *bot* para que, en un futuro, cuando se hagan cambios, se pueda evaluar que las capacidades del bot no se hayan perdido.

3.1.1. Puesta a punto del bot

Es importante que antes de empezar a ampliar las capacidades que ya presenta el *chatbot*, se compruebe que su funcionamiento sea correcto y que no haya fallos fatales que provoquen un comportamiento erróneo. Es por ello que el primer paso será ejecutar el programa, con la finalidad de poner a prueba las capacidades que tiene el prototipo. Cualquier error que se pueda encontrar durante la ejecución deberá ser solucionado para asegurar que el *bot* esté listo antes de empezar a desarrollarlo en mayor profundidad.

Nada más ejecutar por primera vez el programa, se ha encontrado un problema; salta un error y el programa se detiene automáticamente. El mensaje de dicho error, que es bastante extenso, se muestra a continuación.

"geopy.exc.ConfigurationError: Using Nominatim with default or sample `user_agent` "geopy/2.0.0" is strongly discouraged, as it violates Nominatim's ToS <https://operations.osmfoundation.org/policies/nominatim/> and may possibly cause 403 and 429 HTTP errors. Please specify a custom `user_agent` with `Nominatim(user_agent="my-application")` or by overriding the default `user_agent`: `geopy.geocoders.options.default_user_agent = "my-application"``

El error es debido a la configuración del módulo utilizado para gestionar la localización de los hospitales (GeoPy). En el prototipo (sección 7.4 (1)) se utiliza un objeto del módulo GeoPy para la geocodificación (Nominatim), que es el que produce el error; el mensaje dice que hay que modificar uno de los parámetros (*'user_agent'*) que está establecido por defecto y sugiere establecer un valor personalizado como, por ejemplo, '*my-application*'. El código se ha rectificado siguiendo las indicaciones del mensaje, y se ha establecido el parámetro como "TFG Victor", evitando de esta manera la anomalía que surgía al ejecutar el código. Este cambio se encuentra al principio del archivo ([A4.1](#)), cuando se declaran los módulos y las variables globales.

Una vez solucionado el error anterior, se procede a ejecutar el programa de nuevo (esta vez sin problemas iniciales) y a seguir el hilo de la conversación con el bot, para explorar sus características y poder comprobar que todo funciona correctamente. Llegados al punto donde hay que especificar sobre qué enfermedad se quiere que informe el bot, vuelve a saltar otro error. Esta vez el problema surge a causa de la operación de lectura de los datos de la enfermedad almacenados en el archivo JSON; la ruta del archivo al que debe acceder está indicada con una sintaxis incorrecta ("./ProcessedData/"), lo que provoca la detención de la ejecución del *bot*.

"FileNotFoundException: [Errno 2] No such file or directory: './ProcessedData/main_data_melanoma.json"



Este error se debe sencillamente a que el programa no es capaz de encontrar el archivo que se le está especificando. La solución se basa en modificar, en toda parte del código que sea necesaria, el formato de la ruta, para que el programa no tenga problemas a la hora de trabajar con algún archivo. Se ha optado por crear una estructura capaz de encontrar los archivos sin importar la carpeta en la que se almacene el código del programa, siempre y cuando la estructura interna de carpetas del propio proyecto no se modifique. La nueva sintaxis para que se encuentre correctamente el archivo es “..\\ProcessedData\\”.

Solucionado lo anterior, se vuelve a proceder con el hilo de conversación del bot, y otro error surge en el mismo punto de antes. Esta vez se localiza dentro de la función encargada de leer la información, tanto de enfermedades como de hospitales, de los archivos con formato JSON utilizados para almacenar todos los datos; la función se llama “*read_book()*”, del archivo “*utils.py*” ([A4.2](#)), y el mensaje de error que aparece es el siguiente.

“UnicodeDecodeError: 'charmap' codec can't decode byte 0x9d in position 1093: character maps to <undefined>”

El mensaje indica que hay un problema con la codificación en la función “*open()*” que se utiliza dentro de “*read_book()*”; cuando se intenta leer un archivo JSON, el programa es incapaz de descodificar los datos, y por eso hay que cambiar la configuración de la función, variando el parámetro ‘*encoding*’, que tiene establecido por defecto una codificación ‘*charmap*’, por una codificación ‘*utf-8*’, que soluciona el error. ‘*utf-8*’ es una codificación de caracteres muy extendida a día de hoy.

El siguiente error que se detecta aparece cuando el usuario envía su ubicación para obtener el hospital más cercano de su país. Esto provoca que el programa no funcione correctamente y el *bot* es incapaz de dar la respuesta que debería, devolviendo el mensaje de error que se muestra a continuación.

“Error: reverse() takes 2 positional arguments but 3 were given”

En la función “*findHospitalsCountry()*” ([A4.1](#)), que encuentra la distancia entre los hospitales y la ubicación del usuario (sección 7.4 (1)), el método ‘*reverse()*’ recibe un argumento de más, según el mensaje de error. Este método sirve para encontrar una dirección en función de unas coordenadas geográficas. Según la documentación del módulo GeoPy (5), se debe introducir un único argumento, que debe ser o una tupla con los valores de la latitud y la longitud, “(lat, long)”, o una cadena de texto con dichos valores separados por una coma, “lat,long”. El código del prototipo no sigue ninguna de estas dos estructuras, ya que introduce los dos valores como parámetros separados, motivo por el cual surge un error. La solución optada es la de utilizar como argumento una tupla con los dos valores.

Después de todos los cambios hechos en el programa, no se han detectado más errores graves que provoquen un funcionamiento incorrecto o la detención del bot, pero sí que se han encontrado pequeños detalles en su comportamiento que no son los esperados, y que habría que solucionar antes de continuar con el desarrollo del proyecto.

Cuando se establece una conversación con el bot en español, hay un problema en el momento de enviar la ubicación. El primer mensaje que devuelve el bot no está traducido al español, sigue estando en inglés; no supone ningún problema en el funcionamiento del programa, pero no es adecuado que el bot no sea capaz hablar siempre en el idioma del usuario. El tramo de conversación donde ocurre este problema se muestra en la Figura 2. Se ha comprobado con otros idiomas y el error sigue persistiendo, así que es algo generalizado.

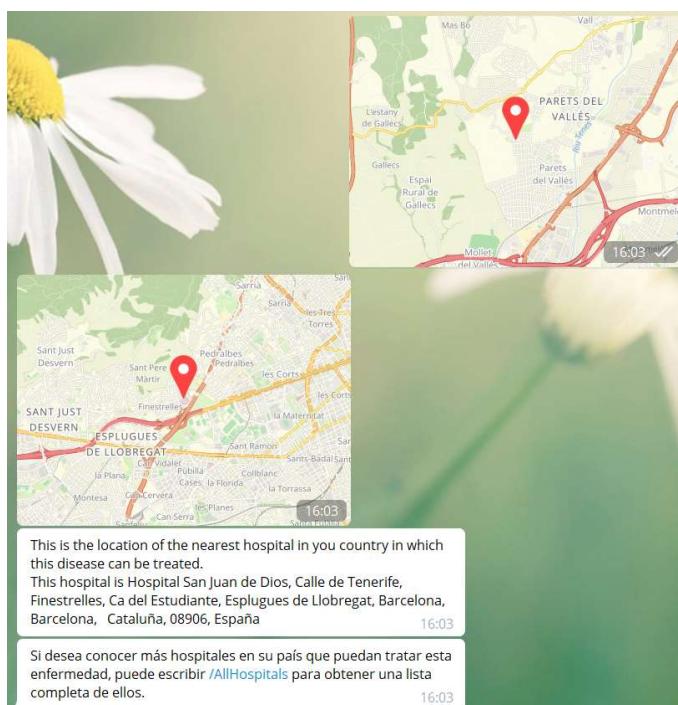


Figura 2. Captura de la conversación con el bot mostrando el error de traducción.

Para solucionarlo hay que acudir al código del programa, concretamente a la función “*giveClosestHospital()*” ([A4.1](#)), que es donde se origina el problema; después de hacer diversas pruebas se ha concluido que el error es debido a la traducción simultánea de dos cadenas de texto, en diferentes idiomas, que se concatenan en una sola variable. El mensaje mal traducido está formado por un texto en inglés más el nombre y dirección del hospital en español; al mezclar ambos textos e intentar traducirlos al español, el módulo de traducción no es capaz de llevar a cabo la operación, al ver que hay una mezcla de idiomas, lo que afecta de forma generalizada, independientemente del idioma establecido inicialmente por el usuario.

La solución aportada es la traducción independiente del texto en inglés, dejando la dirección del hospital en el idioma original (español) y juntando los dos tramos de texto posteriormente al procesamiento de idioma. En un futuro se podría modificar los datos de los hospitales para que la dirección, o al menos parte de esa información, esté también en inglés, pero por ahora no se hará ningún cambio en ese aspecto, ya que no supone un impedimento a la hora de seguir desarrollando el bot, que es lo realmente importante. A continuación, se muestra que el problema ha sido solucionado.

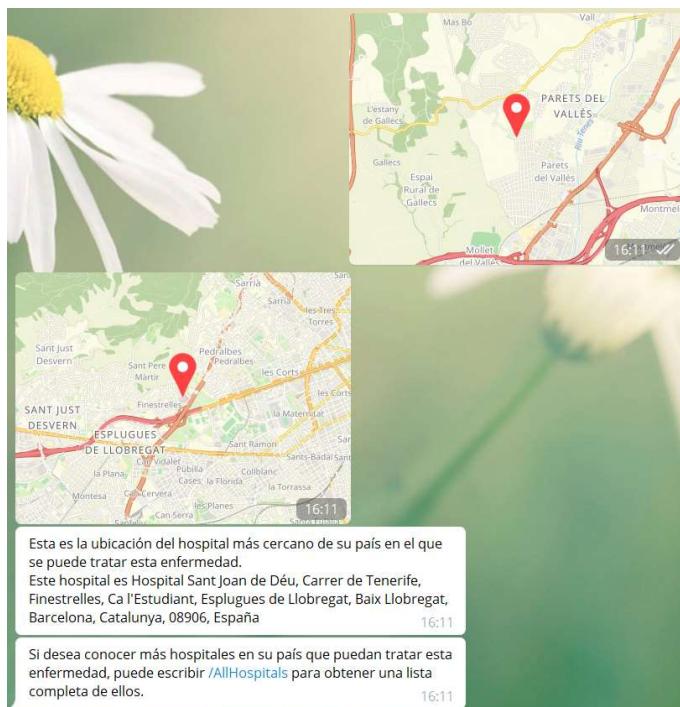


Figura 3. Captura de la conversación del bot con el error de traducción solucionado

El último error significativo se ha encontrado a la hora de consultar información sobre el melanoma; al conversar en inglés o español, el *bot* no da problemas al responder, pero en catalán, en algunos casos, no es capaz de mostrar la fuente de donde ha sacado la información, como, por ejemplo, al preguntar “¿Cómo se diagnostica el melanoma?”. En la Figura 4 se puede ver como en inglés el *bot* sí que da la información completa, pero en catalán no dice la fuente. Al ocurrir esto también salta un error, que no detiene la ejecución del programa, pero da información de por qué ha sucedido. El mensaje del error se muestra a continuación.

“telegram.error.BadRequest”: Can't parse entities: can't find end of the entity starting at byte offset 150”

The screenshot shows a chat interface with two messages. The first message is in English, describing the biopsy process for melanoma diagnosis. The second message is in Catalan, providing the same information. Both messages include a source citation: "Howard L. Kaufman, Janice M. Mehnert - MELANOMA From the section 3.1 How is melanoma diagnosed?" The timestamp for the English message is 19:58 and for the Catalan message is 20:03.

(new moles), any diameter and Evolution of moles. If a skin lesion appears suspicious under the dermatoscope, an excisional surgery should be done for further evaluation. This is an invasive form of examination in which a sample of tissue is taken from the suspicious skin lesion and examined under a microscope by a pathologist to determine its abnormal cellular properties. A representative slice of the skin lesion can be taken (an incision) in order to look at the cells. If the sample turns to be malignant, a wide local excision (WLE) must be performed and ideally accompanied of sentinel lymph node biopsy (SLNB) The depth of the tumour is essential for staging the disease and it is only by taking the entire lesion that the depth can be accurately measured. Sometimes other forms of biopsy such as a punch biopsy is performed to take a sample when the excision is difficult because of the location of the mole (face, hands, toes and genitals).

In case you want more information about it, the information I found comes from this source:
Howard L. Kaufman, Janice M. Mehnert - MELANOMA
From the section 3.1 How is melanoma diagnosed?

bony, uniformitat del color, De novo (mols nous), qualsevol diàmetre i evolució dels talps. Si una lesió cutània apareix sospitosa sota el dermatoscopi, s'hauria de fer una cirurgia per escisió per a unaavaluació posterior. Es tracta d'una forma invasiva d'examen en què es pren una mostra de teixit de la lesió cutània sospitosa que un patòleg examina al microscopi per determinar les seves propietats cel·lulars anormals. Es pot prendre una ilesca representativa de la lesió cutània (una incisió) per mirar les cèl·lules. Si la mostra es converteix en maligna, s'ha de realitzar una extensa extirpació local (WLE) i acompanyada idealment de la biòpsia del gangli limfatíic sentinel·la (SLNB), es pot mesurar amb precisió la profunditat. De vegades es realitzen altres formes de biòpsia, com ara una biòpsia per punxada, per prendre una mostra quan l'excisió és difícil a causa de la localització del talp (cara, mans, dits dels peus i genitals).

Figura 4. Capturas de la conversación del bot en inglés (izquierda) y en catalán (derecha), en respuesta a la

pregunta “¿Cómo se diagnostica el melanoma?”.

El error, que tiene lugar en el núcleo del programa (función “*onMessage()*” de [A4.1](#)), se origina cuando se intenta mostrar el mensaje con la fuente de información. El texto que se utiliza para enviar el mensaje está escrito en formato Markdown, para que, al ser mostrado en el chat, ciertas partes del texto estén en negrita o cursiva. Después de hacer varias pruebas, se ha podido comprobar que el error se encuentra exactamente después de la traducción al catalán; la parte de texto que debería ir entre comillas ha perdido la comilla del principio, y esto provoca el error al intentar mostrar el mensaje en formato Markdown, quedando el texto incompleto para este formato, al no cumplir con las reglas de sintaxis necesarias.

La solución por la que se ha optado es similar a la que se ha implementado en el error de traducción, esto es, traducir por separado los diferentes elementos del texto que se debe mostrar al usuario y, después, adaptarlo al formato Markdown con los distintos símbolos de puntuación necesarios. De esta manera se evitará que el error pueda aparecer sea cual sea el idioma que el usuario utilice con el *bot*.

The screenshot shows a message in Catalan from the bot, correctly formatted with the source information. The message reads: "En cas que desitgeu més informació al respecte, la informació que he trobat prové d'aquesta font: Howard L. Kaufman, Janice M. Mehnert - MELANOMA Des de la secció 3.1 Com es diagnostica el melanoma?". The timestamp is 17:00.

Figura 5. Captura de la conversación con el bot en catalán, mostrando correctamente el mensaje para informar sobre la fuente de información.

3.1.2. Plantilla de evaluación

Una vez se ha puesto a punto el *chatbot*, arreglando los errores que se han encontrado al ejecutar el programa y evaluar sus características, se pretende realizar una agrupación de una serie de preguntas y líneas de conversación con el *bot*, para dejar un registro de su comportamiento, considerándose este como correcto. En un futuro, a medida que se vayan haciendo cambios en el programa, esta plantilla servirá de referencia para evaluar que las capacidades del *bot* no se hayan perdido o que su comportamiento no sea incorrecto, convirtiéndose así en una guía y referencia para el desarrollo del proyecto.

La plantilla de evaluación se realizará en el [Anexo B](#), ya que se pretende ampliarla en un futuro con las nuevas funcionalidades y posibles líneas de conversación, así que no tendría sentido añadir eso a esta altura del proyecto, cuando aún no se ha desarrollado por completo. Inicialmente se ha añadido tres líneas de conversación distintas, una en cada idioma, que busca explorar todas las capacidades generales del *bot*, focalizando en la consulta de información del melanoma ([B1](#)). La principal y más extensa se desarrollará en inglés, que es el idioma con el que se gestiona la información interna, y después se realizará un par de pequeñas líneas de conversación en catalán y español para comprobar que las traducciones a otros idiomas funcionan sin problemas.

Más adelante, una vez se haya terminado el desarrollo del proyecto y se haya implementado correctamente todas las modificaciones y ampliaciones de funcionalidades, se añadirán nuevas líneas de conversación a la plantilla de evaluación, con el fin de dejar registrado el comportamiento de los cambios realizados en la nueva versión del *chatbot*.

3.2. Versión 1

Una vez acabados los cambios fundamentales para el correcto funcionamiento del prototipo, se procederá a alcanzar una versión significativa del programa. En este bloque se concentrarán los cambios que tendrán un impacto más relevante sobre el *bot*; se buscará ampliar la base de datos de enfermedades, esto es, aumentar el número de enfermedades sobre las que el *chatbot* podrá informar, añadiendo información veraz e interesante de cara a los usuarios.

Debido al crecimiento de información que experimentará el *bot*, será necesario optimizar el mecanismo de lectura de datos para asegurar que este proceso no dependa más de archivos locales; se intentará además que sea lo más rápido y eficiente posible. Esto se realizará previamente a la ampliación de la base de datos, ya que será más sencillo trabajar con una cantidad menor de información.

Como se verá más adelante en el transcurso del proyecto, el sistema de búsqueda de resultados frente a las consultas de los usuarios tiene un rendimiento muy deficiente, y será necesario mejorarlo con la finalidad de mejorar la experiencia conversacional del *chatbot*. Esto es importante debido al objetivo último de esta herramienta; la información médica contrastada suele ser difícil de encontrar por parte de pacientes y familiares, así que es importante que se implemente un buen sistema de respuesta capaz de resolver las dudas de aquellos que lo necesitan. Antes de ampliar las enfermedades se realizarán estos cambios, buscando una mejora significativa respecto al prototipo.

3.2.1. Cambio lectura de datos

Actualmente, el mecanismo que se emplea en el programa para leer los datos de las enfermedades es una lectura de unos archivos en formato JSON, que contienen los datos en una estructura de diccionarios de Python. Estos datos se cargan en el programa y se almacenan en una lista que se utiliza para acceder, cuando sea necesario, a los distintos diccionarios que contienen la información de las enfermedades o los hospitales.

La información de la que se disponga crecerá considerablemente, lo que supondrá un problema para el mecanismo actual; por cada nueva enfermedad que se plantea añadir, serían necesarios dos archivos JSON más, uno para la información y otro para las direcciones de los hospitales. Esto sería un problema, dado que se necesitaría una cantidad de archivos muy elevada para almacenar toda la información que necesita el *bot*, y si se planteara unificar todos los datos en un solo archivo, el tiempo que tardaría el programa en ejecutarse iría aumentando cada vez más, ya que la velocidad de lectura de archivos JSON, como se verá más adelante, no es demasiado elevada comparado con otros mecanismos. Además, estos archivos han de ser almacenados localmente en el ordenador desde el cual se ejecuta el programa, lo que ocasionaría problemas si se quisiera subir el programa a un servidor que mantenga el *chatbot* en constante ejecución.

Para solventar estos problemas que se plantean, hay diversas alternativas que pueden asegurar que el programa no sufra en términos de eficiencia y accesibilidad en un futuro. Después de barajar opciones, se ha decidido utilizar Redis (5), un almacén de estructura de datos utilizado habitualmente como base de datos, entre otras, conocido por su velocidad y sencillez. La información se almacena en un formato idéntico al de un diccionario de Python, lo que encaja con la estructura de los datos que se tienen del melanoma y los hospitales, lo que debería facilitar aún más su uso. Además, Redis trabaja con servidores y, a pesar que para el desarrollo de este *bot* se trabajará con un servidor local, tiene la capacidad para implementar un acceso remoto a los datos, sin necesidad de archivos locales almacenados en ningún dispositivo concreto.

Para implementar Redis se ha decidido buscar diferentes propuestas a poner en práctica, para intentar conseguir un funcionamiento lo más eficiente posible y que, sobre todo, no suponga un atraso en velocidad respecto a la mecánica que actualmente está implementada. Cada propuesta tendrá dos fases, una de preprocesamiento de los datos, para subir la información de los archivos JSON a Redis, y otra de lectura de los datos dentro del *chatbot*. Además, se ejecutará el programa con cada propuesta implementada, para comprobar que el comportamiento del *bot* no presente anomalías que antes no existían, comparando los hilos de conversación de la nueva versión con la plantilla de evaluación desarrollada en el [Anexo B](#). Para realizar la fase de preprocesamiento se ha modificado el archivo Python que se utilizaba en el prototipo (sección 7.6 (1)), adaptándolo a las necesidades actuales ([A4.3](#)).



3.2.1.1. Propuesta 1

En un mismo servidor de Redis se dispone de varias bases de datos; la primera idea consiste en almacenar la información del melanoma en una base de datos (*database 0*) y los hospitales en otra (*databases' 1*), siguiendo una estructura similar a la original de los archivos JSON. Dado que en Redis no se pueden almacenar listas directamente, y los datos que se tienen están almacenados en una lista, se debe modificar ligeramente la estructura de estos a la hora de guardarlos en las bases de datos.

Primero de todo, utilizando dos variables, se cargan las bases de datos de Redis como se indica en la documentación (6), para posteriormente almacenar la información en ellas. Los archivos en formato JSON son leídos para extraer los datos; estos son tratados para obtener un formato válido para las bases de datos. Por último, se cargan en Redis los datos procesados mediante *pipelines* que sirven para acelerar el proceso.

De esta manera se consigue almacenar toda la información sin mucho problema ni esfuerzo. La estructura resultante que los datos presentan dentro de Redis (Figura 6), es muy similar a la original, pero substituyendo la lista que ordenaba los diccionarios por otro diccionario cuyas claves están numeradas, empezando por el cero. La información del melanoma queda numerada como ‘data0’, ‘data1’, etc. y los hospitales como ‘hospital0’, ‘hospital1’, etc.

```
# Database 0
{
    "data0": {"title": "1.1 Human skin: Brief overview", "text": "The skin constitutes (...)", "URL": "Howard L. Kaufman (...)"},
    "data1": {"title": "1.2 What is cancer?", "text": "Cells are the building blocks (...)", "URL": "Howard L. Kaufman (...)"},
    "data2": {"title": "1.3 What is melanoma?", "text": "Melanoma skin tumours appear (...)", "URL": "Howard L. Kaufman (...)"},
}

# Database 1
{
    "hospital0": {"lat": 41.3843547, "lon": 2.101905315491, "name": "Hospital Sant Joan de Déu Barcelona", "country": "España"},
    "hospital1": {"lat": 38.3632372, "lon": -0.48591021612, "name": "Hospital General Universitario de Alicante", "country": "España"},
    "hospital2": {"lat": 39.4434289, "lon": -0.37604040198, "name": "Hospital Universitari i Politècnic La Fe", "country": "España"},
}
```

Figura 6. Estructura de los datos almacenados en las bases de datos de Redis para la propuesta 1 de implementación.

Una vez realizado lo anterior, los datos ya pueden ser leídos a través del servidor de Redis, dejando de ser necesario trabajar con archivos JSON. El objetivo es que los datos que se lean se adapten a la misma estructura que se tenía originalmente; de esta manera se evitará hacer ningún cambio en el código del programa principal, que ya está diseñado para trabajar con una estructura de datos concreta. Para conseguirlo se ha modificado el comportamiento de la función ‘*read_book()*’ ([A4.2](#)).

En esta reinterpretación de la función, el parámetro de entrada ha variado respecto al diseño original; debe pasarse la base de datos de Redis correspondiente al bloque de datos que se quiera leer (información o hospitales), que será previamente cargada en una variable como se ha mencionado antes. Mediante una sentencia *for* se va leyendo cada entrada de datos y se extrae el diccionario que contiene la información, almacenándolo en una lista, consiguiendo así la misma estructura que se obtenía al principio. Con estos cambios listos se puede proceder a ejecutar el *chatbot* y comprobar que funcione correctamente.

Al ejecutar el programa para comprobar su funcionamiento, aparece una discrepancia respecto a la plantilla de evaluación. El comportamiento erróneo se muestra en la Figura 7. Cuando se hace la pregunta “What is melanoma?” al bot, su respuesta es distinta a la de la plantilla, además de ser incorrecta.

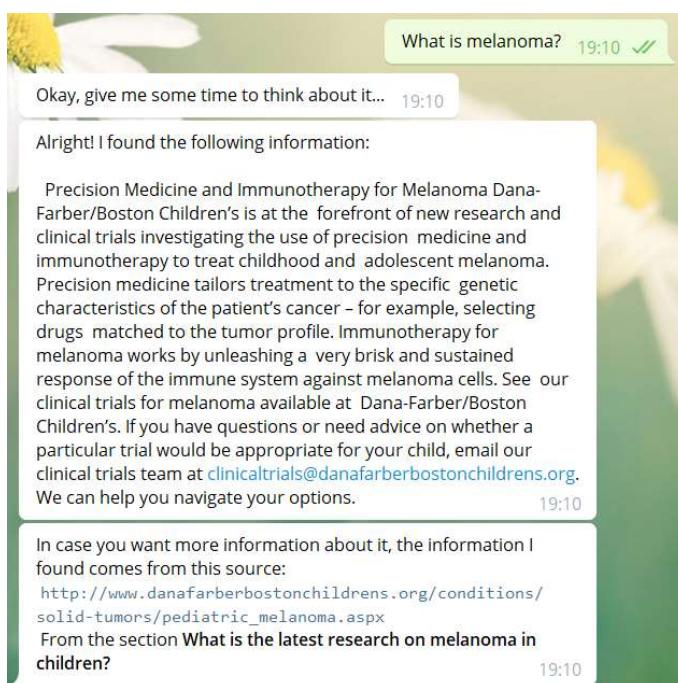


Figura 7. Captura de la conversación con el bot. Se pregunta qué es el melanoma, pero responde de manera incorrecta.

Tras estudiar la posible fuente del cambio de comportamiento, se ha conseguido averiguar lo que ocurre. El programa no actúa de forma incorrecta, pero ha variado en un aspecto en concreto que ha provocado este cambio. La clave de esta situación se encuentra en el cálculo de las similitudes entre la pregunta o consulta que hace el usuario y los datos que tiene el bot. Por cómo está diseñado el prototipo (sección 7.3 (1)), la similitud se calcula entre las palabras clave de la pregunta y las palabras del título de cada fragmento de información sobre el melanoma, guardando solo la similitud más elevada entre cada pareja de palabras; esto tiene un efecto negativo en la eficiencia del bot, ya que,

de las palabras de los títulos de la información, únicamente tendrán relevancia un número igual al de las palabras clave. En el caso de la pregunta “What is melanoma?”, se extraen todas las palabras como clave, dando un total de tres palabras, por lo que solo se tendrán en cuenta tres puntuaciones de similitud respecto a los títulos; para el título “What is the latest research on melanoma in children?”, las palabras que obtendrán una mayor similitud serán ‘what’, ‘is’ y ‘melanoma’, puesto que son las mismas tres palabras clave presentes en la pregunta, y el resto de palabras no se tendrán en cuenta en ningún momento. Si ahora se hace la misma comparación con el título “1.3 What is melanoma?” o con “What is childhood melanoma?”, ocurre la misma situación, por lo que, en definitiva, todos estos títulos obtendrán exactamente la misma puntuación de similitud, tal y como se puede ver en la Figura 8. El programa, ante un caso de empate múltiple de similitud máxima, selecciona como mejor coincidencia el primer bloque de información que se encuentra en la lista, que en el caso del prototipo original es “1.3 What is melanoma?”.

```
['is', 'what', 'melanoma'] ['is', 'what', 'melanoma']
1.3 What is melanoma?      What is childhood melanoma?
0.9999999657915127      0.9999999657915127
['is', 'what', 'melanoma']
What is the latest research on melanoma in children?
0.9999999657915127
```

Figura 8. Imagen de tres casos de cálculo de similitud para la pregunta “What is melanoma?”. La primera fila son las palabras clave de la pregunta, la segunda es el título con el que se comparan y la tercera es la puntuación de similitud calculada, que va del 0 al 1. Se puede observar que la puntuación para los tres casos es la misma.

En el prototipo inicial, los datos están ordenados de una cierta forma, pero en esta propuesta, al cargar los datos de Redis, los datos acaban desordenados. Esto tiene un gran impacto cuando se junta con el problema explicado anteriormente, provocando que el primer bloque con mayor similitud haya cambiado de posición. En esta nueva situación, el primer título con la máxima similitud es “What is the latest research on melanoma in children?”, que resulta ser la nueva respuesta que proporciona el bot. Tras todo este análisis, ha quedado claro que el funcionamiento de esta propuesta no es incorrecto, sino que hay un problema de base en el prototipo que se debe solucionar.

3.2.1.2. Propuesta 2

La segunda propuesta tiene un enfoque distinto; se intentará tratar los datos lo menos posible, para que la carga y descarga de estos sea lo más rápida posible. Para lograr esto se utilizará un módulo de serialización y deserialización, que sirve para empaquetar objetos, transformándolos en una serie de bytes, y desempaquetarlos, volviendo a obtener el objeto original; es muy similar al concepto de almacenar los datos en un archivo JSON, pero en este caso se busca que la información esté almacenada en un servidor y no en un archivo local.

Hay muchas herramientas capaces de realizar esta acción, pero la que aquí se baraja es un módulo de Python llamado Pickle (7). Para justificar el uso de este módulo, se comparará su actuación con la de otro módulo que se ha estado utilizando en el prototipo hasta ahora para almacenar los datos en los archivos JSON, el módulo Json (8). Para realizar la comparación se utilizará un test de velocidad que evaluará los dos módulos, utilizando concretamente el archivo JSON que contiene los datos del melanoma. En el lugar de origen del test (9) se puede ver cómo se ha aplicado a varios conjuntos de datos, y los resultados son muy similares a los que se han calculado con el archivo del melanoma (Figura 9).

Name (time in us)	Min	Max	Mean	StdDev	Median
test_speed[pickle]	76.6040 (1.0)	15,867.7180 (17.07)	122.1566 (1.0)	381.7147 (6.22)	84.2550 (1.0)
test_speed[json]	189.8920 (2.48)	929.4620 (1.0)	230.7363 (1.89)	61.3664 (1.0)	201.8220 (2.40)

IQR	Outliers	OPS (Kops/s)	Rounds	Iterations
16.1897 (1.0)	49;1223	8.1862 (1.0)	9139	1
53.9052 (3.33)	450;234	4.3340 (0.53)	3237	1

Figura 9. Test de velocidad utilizando los módulos Pickle y Json sobre el conjunto de datos del melanoma.

Se puede ver como la velocidad del módulo Pickle es, en general, superior, ya que la media (*Mean*) del tiempo de ejecución ha resultado ser menor; el rango intercuartílico (IQR) es más de tres veces menor en el caso de este módulo, indicando que la dispersión del test ha sido poco elevada, siendo así fiable el resto de datos y, por tanto, se puede concluir que la velocidad del módulo es significativamente mayor respecto al anteriormente utilizado en el prototipo. Si a esto se le une el hecho de que se va a almacenar la información en un servidor en línea, sólo se obtienen ventajas al plantear esta propuesta.

Con esta funcionalidad en mente, se pretende cargar los datos como un gran bloque empaquetado, lo que supone varias ventajas, entre ellas, que se pueden almacenar todos los datos sin necesidad de preprocesarlos, solo es necesario guardar el bloque empaquetado en una clave de una base de datos de Redis, lo cual hace que solo sea necesario trabajar con un *database*, facilitando así la comprensión y estructuración de los datos almacenados; esto también solventa el inconveniente de no poder almacenar listas en Redis.

El código utilizado en esta ocasión ha resultado ser menos extenso, además de necesitar solo operaciones muy básicas. Se almacena la información del melanoma, previamente empaquetada, en una clave llamada ‘melanoma_data’, y de igual manera los hospitales se almacenan como ‘melanoma_hosp’, todo ello en el *database* cero, el único que se utiliza. Así se consigue que lo único necesario para utilizar estos datos sea leer los objetos serializados del servidor de Redis y desempaquetarlos, guardándolos en alguna variable dentro del programa del *bot*. Esto se implementa dentro de la función ‘*read_book()*’ comentada anteriormente.

Una nueva reinterpretación de la función es todo lo necesario para hacer efectiva la lectura de los datos. En esta ocasión se necesitan dos parámetros, siendo ‘key’ el nombre de la clave que contiene los datos que se necesitan y ‘r’ la base de datos de Redis, que se cargará al iniciar la ejecución del programa, donde están almacenados dichos datos. De esta forma, solo hace falta indicar si se quiere obtener la información de la enfermedad o de los hospitales, y la función extrae y desempaquetá directamente los datos.

Al ejecutar el programa con esta implementación, no se ha detectado ningún error de funcionamiento y el comportamiento coincide con la plantilla de evaluación. A diferencia de la propuesta anterior, en esta, los datos mantienen el mismo orden dentro de la lista que tenían en el prototipo original, manteniendo así las mismas respuestas, a pesar del problema de los empates de similitud.

3.2.1.3. Análisis final

Tras haber diseñado e implementado las dos propuestas anteriores para la lectura de los datos con Redis, es necesario analizar cuál de ellas es más eficiente y supone una mejora mayor respecto al sistema original. El indicador de referencia, dado que ha sido uno de los parámetros que ha impulsado el cambio en la lectura de datos, será la velocidad de ejecución, esto es, cuanto menor sea el tiempo que tarda el programa en ejecutar la función de lectura de datos, mayor será la eficiencia de la propuesta diseñada.

Para calcular el tiempo de ejecución del programa, se ha establecido un contador de tiempo al inicio de la llamada de la función '*'read_book()'*', que deja de contar cuando se sale de esta. Con esto se consigue medir en igualdad de condiciones el tiempo que tarda cada propuesta en ser ejecutada. En la Tabla 1 se recoge, para cuatro casos de ejecución del programa, los tiempos del prototipo original y las dos propuestas diseñadas, con el fin de comparar con facilidad la eficiencia de todas ellas.

Tabla 1. Comparación del tiempo de ejecución del prototipo original y las dos propuestas desarrolladas.

Original	Propuesta 1	Propuesta 2
0.997 ms	14.935 ms	0.998 ms
1.001 ms	13.96 ms	1.019 ms
1 ms	16.917 ms	0.962 ms
0.999 ms	17.013 ms	0.996 ms

Como se puede observar en la tabla, el tiempo de ejecución de la función original utilizada para leer los datos guardados en archivos JSON se encuentra en torno a 1 milisegundo. Si se compara con el tiempo de ejecución de la primera propuesta, que son unos 15 milisegundos de media, se puede concluir que el cambio realizado no solo no supone una mejora en la velocidad, sino que empeora en este aspecto; el programa se vuelve quince veces más lento a la hora de almacenar la información necesaria sobre el melanoma en un diccionario. Como es comprensible, esta propuesta es descartada dado que no presenta una mejora en la funcionalidad del programa, a pesar de que solventa el problema de tener que almacenar los datos en archivos locales.

Observando el indicador de la segunda propuesta, se ve que el tiempo de ejecución se sitúa en torno a 1 milisegundo, de igual manera que en el prototipo original. Esta implementación no solo no empeora la eficiencia del programa, sino que permite almacenar los archivos en servidores en línea, sin necesidad de disponer de multitud de archivos locales. A pesar de no suponer una mejora directa en la velocidad, tal como se ha observado anteriormente, al incrementar el volumen de datos disponibles

para el *bot* (añadiendo nuevas enfermedades o información adicional de las ya existentes), el rendimiento de la lectura utilizando archivos JSON disminuirá y, por tanto, implementar bases de datos de Redis para la gestión de la información supondrá una mejora a largo plazo. Es por todos estos motivos que la propuesta seleccionada para ser implementada definitivamente sea la segunda.

Durante el diseño y análisis de las propuestas a implementar para la lectura de datos, tal como se ha explicado en la primera propuesta, se ha detectado un bajo desempeño a la hora de encontrar la similitud entre las preguntas de los usuarios y los datos disponibles para el *bot*, obteniendo muchas situaciones donde dicha similitud es la misma para diferentes fragmentos de información. Esto es algo que se debe mejorar, ya que uno de los objetivos del proyecto es conseguir que el bot sea capaz de responder de manera correcta y veraz a las consultas de los usuarios y, por tanto, la mejora del sistema de búsqueda de información será el siguiente paso a realizar.



3.2.2. Mejora del sistema de búsqueda

La ampliación de nuevas enfermedades al *bot* supondrá un aumento significativo de datos e información, que deberá ser tratada de cierta forma; antes de alcanzar ese paso, es importante que las funcionalidades básicas funcionen correctamente y de la manera más eficiente posible, ya que será más fácil trabajar con un volumen de datos menor. El sistema de búsqueda de resultados es una de esas funcionalidades; los datos que proporciona el *bot* a los usuarios deben corresponder con las consultas que recibe, en caso contrario, la funcionalidad básica y fundamental presentará un comportamiento erróneo.

El sistema de búsqueda de resultados y el cálculo de similitudes asociado a este presentan un desempeño muy poco elevado, tal como se ha demostrado anteriormente, siendo necesaria una mejora que permita al *bot* ser capaz de responder a los usuarios con la información correcta. Para la realización de esto, se estudiarán varias propuestas basadas en diferentes mecanismos de comparación y búsqueda de similitudes; estas propuestas corresponderán con los objetivos marcados al inicio del presente documento, buscando así una mejora de la eficacia del programa.

Se implementarán un total de dos propuestas, que serán estudiadas en profundidad en los siguientes apartados. Se buscará implementar dos técnicas de procesamiento del lenguaje natural, en concreto *Word Embedding* y *Sentence Embedding*. Se utilizará la librería Spacy, presente en el prototipo original, para implementar *Word Embedding*; se realizará una modificación del sistema actual de búsqueda, maximizando así su desempeño. Por otro lado, para implementar *Sentence Embedding*, se utilizará BERT, en concreto dos modelos ya preentrenados, llamados ‘WebBertSimilarity’ y ‘ClinicalBertSimilarity’; se implementarán dos modelos que utilizan la misma técnica porque están entrenados de manera distinta, pudiendo así ofrecer diferentes resultados interesantes de analizar y comparar.

Una vez implementados todos los modelos, se buscará una manera de reducir la probabilidad de fallo del *bot* ante preguntas difíciles o respuestas poco acertadas mediante la habilitación de un sistema capaz de devolver al usuario más de una respuesta al mismo tiempo, con el fin de que este sea capaz de escoger la entrada de información que le parezca más acertada respecto a su pregunta inicial. También se barajará la posibilidad de que esta característica dependa de la cercanía entre las puntuaciones de similitud de las respuestas.

3.2.2.1. SpaCy

En el prototipo original del *bot*, se utiliza la librería Spacy (10) para encontrar la similitud entre las consultas de los usuarios y los títulos de los fragmentos de información disponibles, pero su implementación no es correcta ni eficaz; se extraían las palabras clave de la consulta del usuario y se pasaban por una función que realizaba un análisis morfológico, pero todo este proceso no es necesario, ya que la funcionalidad del módulo Spacy que calcula la similitud semántica aplica automáticamente todos los tratamientos que necesita y, por tanto, al realizar modificaciones previas lo único que se consigue es entorpecer el cálculo y reducir la precisión y eficacia del modelo. Para el rediseño de esta funcionalidad, se utilizará la misma librería, pero modificando la implementación de la misma, para obtener un desempeño correcto y más eficaz respecto al prototipo original.

Dentro de la función ‘compute_scores_from_index()’ ([A4.2](#)) se eliminan todos estos tratamientos que se realizaban a las consultas, utilizando directamente el texto original para encontrar las similitudes. De esta manera, el código de la función queda bastante reducido, utilizando directamente la consulta que hace el usuario, sin modificaciones, para realizar el cálculo con todos los datos almacenados, mediante la función ‘get_score()’, encargada de encontrar la similitud semántica de los dos textos.

Una vez realizado el cambio en la función, se pasa a plantear la próxima mejora; en el sistema original, para calcular la mejor similitud, se comparan las consultas de los usuarios con los títulos de los fragmentos de información disponibles para el *bot*, pero esto supone un problema, ya que, en ocasiones, dichos títulos no reflejan bien la información que contiene cada bloque. Para solventar esto, se plantea implementar el cálculo de similitud no solo con los títulos, sino también con el contenido principal de cada bloque de datos; entre las dos puntuaciones de similitud que se obtengan, se elegirá y establecerá la de mayor valor para cada entrada de información. Con estos cambios se conseguirá que las preguntas directas como qué es el melanoma se respondan bien gracias a la comparación con los títulos, y que las preguntas indirectas sobre, por ejemplo, los síntomas o la diagnosis se respondan bien gracias a la comparación con el contenido. Para implementar esto, primeramente hace falta que, en la función ‘compute_scores_from_index()’, cuando se pasa la pregunta y el título por la función ‘get_score()’, se modifique para que se pase todo el bloque de datos, sin separar título y contenido. Para esto se renombrará el objeto ‘index’ por ‘dic’, para dejar claro que se pasa el diccionario entero, con título y contenido, por la función.

En este punto se debe pasar a modificar la función ‘get_score()’, que es donde se concentra el cálculo de las similitudes como tal. Esta función comparaba una a una cada palabra de las consultas con cada palabra de los títulos. Ahora que se ha eliminado la obtención de palabras claves y, por tanto, la separación de los textos en palabras, esto ya no será necesario, ya que se querrá obtener directamente

la similitud entre los textos enteros de las consultas y los títulos. Este cambio simplificará enormemente el código de la función, a la vez que aprovechará al máximo el potencial de la librería Spacy.

Hay que recordar que también se ha planteado la comparación de las consultas con los contenidos asociados a cada título, guardando al final la puntuación de similitud de mayor valor entre las dos calculadas. Por esto y por lo explicado anteriormente, los dos parámetros de la función, ‘keywords’ y ‘title’, se renombrarán como ‘question’ y ‘data’, para conseguir un claro entendimiento de qué tipo de elementos forman dichos parámetros.

Esta nueva implementación actúa tal como se ha explicado anteriormente: compara la consulta hecha por el usuario con el título y contenido del bloque de información que se haya pasado por la función y, entre estas dos puntuaciones de similitud que se obtienen, se devuelve la de mayor valor, estableciéndose así la similitud con ese bloque en concreto; con la llamada reiterada de esta función se acaba consiguiendo la puntuación de similitud con todos los bloques de información.

Adicionalmente, para mejorar el desempeño de la librería Spacy, se ha decidido modificar los textos correspondientes a los títulos y las consultas, previamente al cálculo de similitudes, de tal manera que se eliminan los signos de interrogación. Esta modificación se ha realizado para obtener un cálculo de similitudes más preciso; al eliminar los signos de interrogación, tanto si el usuario ha utilizado uno como si no (a veces la gente, cuando escribe en chats, omite signos de puntuación), la comparación de esas consultas con los títulos será la misma. Esto es importante que sea así, ya que la herramienta para calcular la similitud distingue entre interrogación y afirmación, esto es, si una consulta estuviera escrita sin signo de interrogación, como, por ejemplo, “*What is melanoma*”, a la hora de compararla con el título “*What is melanoma?*”, se obtendría una puntuación de similitud distinta que si la consulta fuera escrita con signo de interrogación, a pesar de que el usuario estaría preguntando lo mismo. Eliminando los signos de interrogación se elimina esta diferencia entre ambos casos, obteniendo así un mismo comportamiento, como cabría esperar. También se han eliminado los números de los títulos, ya que algunos de ellos presentan numeración de apartados que no aportan ninguna información útil y que solo disminuyen las puntuaciones de similitud obtenidas de manera innecesaria.

3.2.2.2. WebBertSimilarity y ClinicalBertSimilarity

Dada la poca capacidad que presenta el modelo de Spacy para entender el contexto del texto, se necesita otro modelo que sí sea capaz de hacer esto, con el fin de mejorar el sistema de búsqueda del *bot*; los modelos de BERT tienen esta capacidad, y por eso se ha elegido implementar dos en concretos ya preentrenados (11). De igual manera que en el apartado anterior, se irán explicando los elementos añadidos y los cambios necesarios respecto a la implementación con Spacy.

En este caso en concreto, dado que la estructura de procesamiento de similitudes ya está formada, y lo único que variará respecto a Spacy es la herramienta que se utilizará para el cálculo, no será necesario modificar en gran medida el código; se substituirán las líneas de código que implementan Spacy por otras que hagan lo mismo, pero con los modelos BERT. El primer paso, al igual que con Spacy, es cargar los modelos en forma de variable, previamente a la utilización de estos.

Como se ha comentado antes, el único cambio que se realizará dentro del código será la substitución de la manera de obtener las puntuaciones de similitud. Para esto, dentro de la función ‘*get_score()*’, simplemente se debe utilizar el método ‘*predict()*’ de los modelos, que devuelve una puntuación de similitud en función de los dos textos que se introduzcan; dichos texto, que en este caso son la consulta del usuario y el título o contenido de los bloques de información, se pueden introducir directamente sin necesidad de realizar ningún tratamiento previo, lo que simplifica mucho el código.

Así, sin modificar en ningún momento la estructura ni el funcionamiento que presentaba previamente el *bot*, se han implementado los modelos de BERT, que proporcionan respuestas que tienen en cuenta el contexto de los textos, que puede ser muy útil a la hora de comparar las consultas de los usuarios con el contenido de los bloques de información.

3.2.2.3. Sistema de respuesta múltiple

A pesar de los modelos implementados en los apartados anteriores, no es posible asegurar que la calidad en la búsqueda de resultados sea lo suficientemente elevada, sobre todo cuando se realizan consultas muy extensas o poco específicas, ya que estos modelos están preentrenados para presentar un funcionamiento sencillo, perdiendo así la ventaja de implementar un modelo personalizado que sea capaz de tratar con la mayor rigurosidad posible los datos. Esto supondría una tarea mucho mayor, dado que se tendría que diseñar a fondo un sistema de NLP.

Con el fin de mitigar lo máximo posible la falta de precisión ante consultas difíciles de responder, se ha diseñado un sistema de respuesta múltiple utilizando una de las características que presenta la API de Telegram (12); se ha utilizado una funcionalidad que permite la creación de teclados personalizados que se muestran al usuario para elegir una respuesta predeterminada. Con esta herramienta en mente, se pretende sugerir al usuario un total de tres entradas de información, que serán las tres de mayor similitud calculada por los modelos de NLP. De esta manera, si por algún motivo el bloque de datos con mejor puntuación no es el más adecuado para responder, se podrá visualizar y escoger entre otras dos opciones más que quizás se ajusten mejor a las necesidades de la consulta.

Para implementar lo descrito antes es necesario modificar el código de varios archivos, en concreto las funciones “*process_message()*” ([A4.2](#)) y “*onMessage()*” ([A4.1](#)). En el caso de la primera mencionada, se ha variado el procesamiento de la salida de la función que se realizaba anteriormente (sección 7.3 (1)); se ha pasado de devolver directamente la entrada de información con la mayor similitud respecto a la consulta a devolver una lista con los índices (posición en la lista de diccionarios con información) de los tres bloques de datos que han obtenido las tres puntuaciones más altas de STS. Habiendo obtenido esta lista, dentro de la función “*onMessage()*”, se procede a enviar un mensaje al usuario dando a elegir entre las tres opciones, mostrando un teclado que contiene tres botones con el nombre del título de cada una de las entradas de datos. Una vez el usuario ha elegido la opción que desea, se devuelve inmediatamente la información del bloque seleccionado, utilizando el mismo formato que se utilizaba con anterioridad. Para realizar esto se ha ampliado el bloque condicional principal que se utiliza en la función “*onMessage()*” para contemplar la situación en la que el *chatbot* ha enviado las tres posibles respuestas y está esperando a recibir el nombre del título del bloque de datos que debe utilizar para informar; usando un *flag* que indica el estado de este proceso (en curso o no), se implementa una nueva rama en la estructura de árbol de decisiones del *bot* que realiza el proceso de enviar la información en función de la opción elegida en el teclado. Se baraja la opción de eliminar el teclado una vez se ha utilizado, pero también es interesante que se mantenga abierto, ya que así el usuario puede consultar sobre el resto de opciones si desea informarse también de ellas.

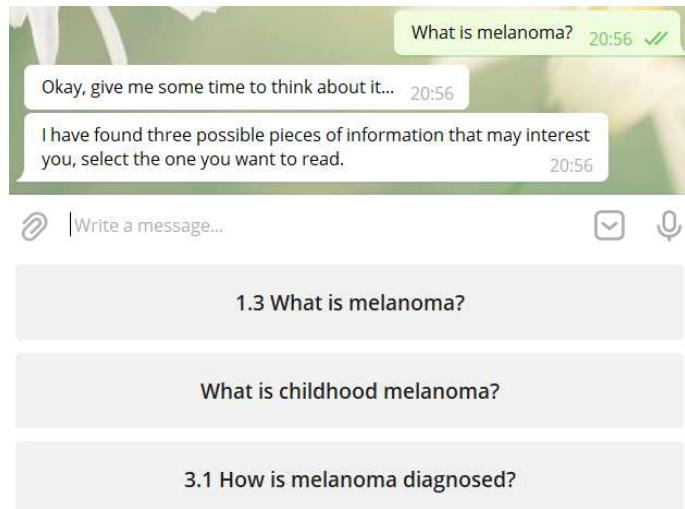


Figura 10. Captura de la conversación con el bot, mostrando al usuario los tres bloques de información con mayor coincidencia.

El resultado obtenido implementando el diseño anterior se refleja en la Figura 10, donde se puede observar como el chatbot, ante la consulta de qué es el melanoma, ahora devuelve un mensaje notificando al usuario de que ha encontrado tres posibles piezas de información que podrían interesarle; entre las opciones se encuentra, en primer lugar, la respuesta con mayor similitud, que es la que daba anteriormente cuando no existía este sistema, y además propone otras dos opciones que podrían interesar al usuario. La segunda de ellas (“¿Qué es el melanoma infantil?”) es una información muy parecida a la primera, pero especifica más el tipo de melanoma, lo que podría interesar más al usuario en el caso de estar buscando una información más específica, sirviendo esto como prueba de la utilidad de este sistema de selección de respuesta múltiple.

3.2.2.4. Características adicionales

Una vez implementados los diferentes modelos de procesamiento del lenguaje para el cálculo de la similitud semántica y el sistema de respuesta múltiple, se plantean características adicionales a implementar en el *bot* para realizar un uso más eficiente de todo lo desarrollado en este apartado.

Primero de todo, ya que se ha implementado el uso tanto de Spacy como de los modelos de BERT sería un error eliminar ninguno de ellos, por lo que se ha decidido diseñar un sistema que permita seleccionar el modelo a utilizar, pudiendo así cambiar cómodamente entre uno u otro en función del que se quiera utilizar. Para esto se ha añadido un parámetro a la función “*process_message()*” ([A4.2](#)) que se llamará “*model*”, y que podrá tomar tres valores distintos ('Spacy', 'WebBert' y 'ClinicalBert'), en función del modelo que se quiera establecer. Este parámetro se enviará a través de la función “*get_score()*” y permitirá, mediante el uso de una estructura condicional, ejecutar el cálculo de la STS con un modelo u otro. De esta manera se consigue variar la herramienta para el cálculo de la similitud de forma muy sencilla, sin necesidad de variar el código del programa en ningún momento, excepto para establecer un valor concreto al parámetro añadido.

Por otro lado, el sistema de respuesta múltiple, a pesar de ser una buena herramienta para mitigar un mal resultado en la búsqueda de respuestas, puede llegar a ser un inconveniente si se ejecuta el cien por ciento de las veces, ya que el usuario esperará una conversación lo más fluida posible con el *chatbot*. Es por ello que se ha diseñado un sistema para decidir, en función de las tres respuestas con más similitud, si el *bot* debe responder directamente con el mejor resultado o si, por el contrario, debe utilizar el sistema de respuesta múltiple. Esto se realiza mediante un simple condicional dentro de la función “*process_message()*”, que, en función de la distancia entre los valores de similitud de las dos mejores respuestas, se decide qué acción llevar a cabo. Si el condicional establece que se debe enviar múltiples respuestas, la función devolverá la lista con los índices de los tres mejores resultados, como se ha explicado en el apartado anterior, pero si solo se decide enviar una única opción, se enviará una lista con un único índice disponible. Dentro de la función “*onMessage()*” ([A4.1](#)), cuando se realiza el procesamiento libre del texto, se ha añadido un condicional que comprueba el tamaño de la lista que es devuelta por la función “*process_message()*” y, en función de esto, se decide si cambiar el estado del *flag* que ejecuta el sistema de respuesta múltiple en el siguiente procesamiento de texto o si, por lo contrario, se prosigue con el funcionamiento habitual de enviar una respuesta directa.

Todo esto se ha implementado bajo la lógica de que si la respuesta con mayor STS dista mucho de la segunda (es claramente un mejor resultado para la consulta), no será necesario sugerir varias opciones, mientras que, si las primeras respuestas son muy similares entre sí, es interesante sugerir al usuario varias posibilidades para que él elija. Así se consigue aumentar la fluidez del *chatbot* a la par que se tiene en cuenta la posibilidad de que la respuesta encontrada no sea muy acertada.



De igual manera que con la selección de modelo para el cálculo de STS, se ha implementado un parámetro adicional a la función “*process_message()*”, llamado “*multiple*”, que permite establecer si se desea que el *bot* realice siempre un proceso de respuesta múltiple o si, por el contrario, se quiere comprobar si es necesario que se envíen diversas opciones al usuario o si puede devolver un resultado directo. El valor de la distancia mínima asociada a la toma de decisión de este proceso se ha establecido como un valor fijo, pero puede ser fácilmente variado para conseguir que dependa de otros factores, que será algo a tener en cuenta en futuras actualizaciones del *bot*, ya que los rangos de valores en el cálculo de la STS por parte de los diferentes modelos implementados no son los mismos y, por tanto, se deberá aplicar una normalización de los valores o algún otro tipo de tratamiento que solucione esto. También es interesante barajar la posibilidad de añadir un parámetro adicional más a la función que permita establecer el valor que se quiere utilizar a la hora de comprobar la distancia entre los dos mejores resultados.

Todo lo expuesto aquí, concretamente la creación de parámetros para la función “*process_message()*”, tiene el potencial de enfocarse a una aplicación más general de todos estos sistemas, esto es, se podría crear nuevos comandos o líneas de conversación que permitan al usuario decidir si desea que la búsqueda de información sea más precisa, utilizando BERT, o si prefiere que sea más rápida, utilizando Spacy (como se verá en el siguiente apartado, el tiempo de ejecución del modelo de Spacy es menor que en el caso de utilizar BERT).

3.2.2.5. Análisis final

Para poder cerrar el desarrollo de este apartado, se debe antes realizar un análisis del impacto resultante de todas las nuevas medidas implementadas en el *chatbot*. Para abordar esto se buscará comparar en ciertos aspectos el rendimiento de los diversos modelos de procesamiento del lenguaje que se han utilizado para realizar el cálculo de la similitud semántica entre las consultas de los usuarios y la información almacenada de la enfermedad.

Primero de todo, para comprobar que la precisión de los modelos BERT sobre Spacy es superior, se realizará un cálculo de la correlación obtenida al aplicar estos modelos sobre un conjunto de datos concreto. Estos datos estaban previamente preparados para evaluar módulos de cálculo de similitudes semánticas, y se ha utilizado un test preparado específicamente para encontrar la correlación de los resultados de cada modelo. El conjunto de datos y el archivo que contiene el programa que ejecuta el test se encuentran en el archivo adjunto a este documento ([A3](#)). El resultado de aplicar el test se puede observar en la Figura 11; la correlación utilizando Spacy es apenas de 0.5, mientras que los modelos BERT, al llegar a más de 0.8, obtienen un valor significativamente mayor, lo cual indica que la precisión de estos es mayor respecto a Spacy.

*		*
Correlación		
*	Spacy	0.507
*	WebBert	0.861
*	ClnBert	0.837

Figura 11. Correlación calculada de los diferentes modelos aplicados al *Test Gold*.

Para comprobar la teoría mostrada antes, se han hecho pruebas para encontrar un caso concreto de consulta que el *bot* no es capaz de responder correctamente con Spacy, pero sí con BERT. A pesar de estar adelantando acontecimientos, todo esto se comprobará sobre la enfermedad añadida más adelante (apartado [3.2.3](#)), la ELA, porque en su conjunto de datos se encuentra un caso muy específico donde se ve claramente la diferencia al utilizar un modelo u otro.

De igual manera que se hizo al principio del proyecto, se le consulta al *bot* algo muy básico sobre la enfermedad, se pregunta qué es la ELA. Para comunicarse con el *bot* se utiliza concretamente la pregunta “*What is sclerosis?*” (Figura 12); al consultar con el modelo de Spacy, la respuesta no tiene sentido, ya que devuelve un bloque de información con el título “*Getting a Second Opinion*”, mientras que con el modelo ‘ClinicalBertSimilarity’, la respuesta es la esperada, devolviendo “*What is ALS?*”. A pesar de estar escrito solo parte del nombre de la enfermedad en el mensaje (*sclerosis*), utilizando BERT el *bot* ha sido capaz de devolver información que corresponde con la consulta realizada, mientras que con Spacy no ha sido capaz de encontrar la entrada correcta.

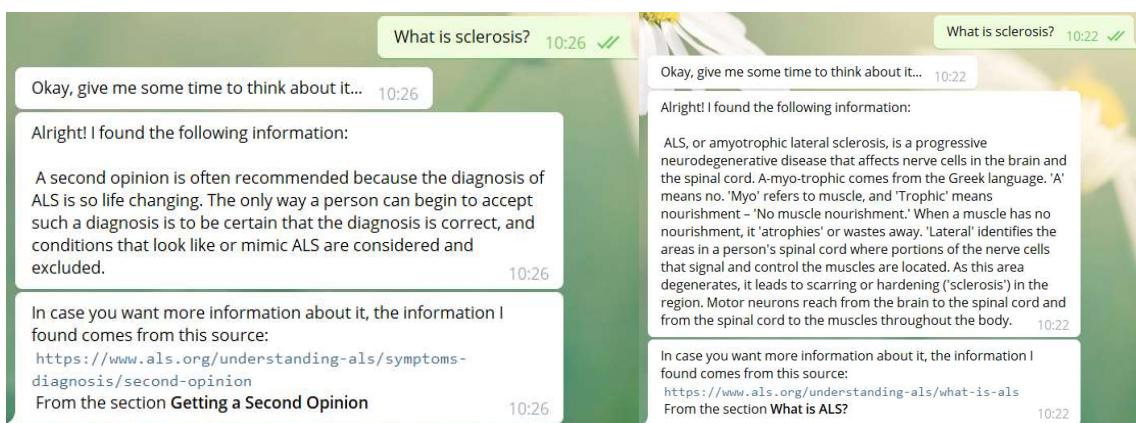


Figura 12. Captura de la conversación con el bot. Se puede observar la respuesta a la pregunta “*What is sclerosis?*” con el modelo de Spacy (izquierda) y con los modelos BERT (derecha).

Después de observar que la eficacia y precisión se ven incrementadas gracias a la implementación de los modelos BERT, se pasará a analizar el tiempo de ejecución de cada uno de los modelos, que es otro punto importante a tener en cuenta, ya que no solo la precisión es necesaria, sino también la velocidad, para poder brindar a los usuarios del *chatbot* una experiencia lo más fluida posible.

Para esto se ha sometido el *bot* a unas pruebas muy parecidas a las realizadas en el apartado [3.2.1.3](#); se introducen contadores de tiempo al inicio y final de la función encargada de realizar el cálculo de las similitudes y de devolver la información más relevante (“*process_message()*”, [A4.2](#)) y, realizando siempre la misma consulta, se podrá ir registrando el tiempo de ejecución. Con esto se consigue medir el tiempo de este proceso en igualdad de condiciones para todos los modelos aplicados. Para estas pruebas se ha desactivado por completo el sistema de selección de respuestas múltiples, ya que solo interesa analizar el comportamiento en el caso de recibir una única respuesta. Se utilizarán los datos del melanoma para realizar las consultas, preguntando siempre qué es el melanoma (“*What is melanoma?*”).

Tabla 2. Comparación del tiempo de ejecución de los tres modelos de cálculo de similitudes para la pregunta “What is melanoma?”.

Spacy	Web Bert	Clinical Bert
3,21 s	34,89 s	35,25 s
3,14 s	35,3 s	35,11 s
3,15 s	34,6 s	34,81 s
3,09 s	35,13 s	34,84 s

Una vez ejecutado el *bot* un total de cuatro veces para cada modelo, se han recogido los datos obtenidos (Tabla 2) para realizar su análisis. Como se puede observar, el tiempo de ejecución con Spacy es bastante reducido, de tan solo 3 segundo aproximadamente, mientras que, con los modelos de BERT, el valor obtenido es alrededor de 35 segundo; el primero de los modelos llega a ser hasta 11 veces más rápido al realizar el proceso de cálculo de STS. Esto es debido en parte a que ahora el cálculo de similitudes se realiza también con los contenidos de los bloques de datos y, al ser textos bastante extensos, utilizando BERT se eleva mucho el tiempo de ejecución, ya que los procesos de este modelo son más lentos al tener en cuenta el contexto de los datos.

Después de haber analizado tanto la eficacia como la velocidad de las implementaciones de STS realizadas, se puede concluir que utilizando los modelos de BERT se puede conseguir unos resultados más precisos, y el bot no debería depender tanto del sistema de respuesta múltiple, que beneficiaría la fluidez, mientras que, si se utiliza Spacy, la velocidad del *bot* aumenta considerablemente, haciendo la conversación con el *bot* mucho más fluida, y puede compensar un poco su falta de precisión gracias a la respuesta múltiple. Esto significa que es interesante tener en cuenta, al menos por ahora, ambos modelos, ya que se puede depender de una herramienta o de otra en función de las necesidades concretas que pueda tener cada usuario. Aun así, hay que remarcar que todavía queda mucho margen de mejora, pudiéndose mejorar mucho el sistema de búsqueda de respuesta, ya sea optimizando los modelos actuales, sobre todo los de BERT, que son los que tienen más potencial, o creando modelos personalizados entrenados específicamente para el *chatbot*, utilizando técnicas de *Machine Learning* y *Deep Learning*.

3.2.3. Ampliación de enfermedades

Una vez implementados diversos modelos para mejorar el sistema de búsqueda de respuestas, se considera que las funciones básicas del *chatbot* han alcanzado un nivel adecuado para poder continuar con el desarrollo del proyecto; en este punto se puede proceder a añadir nuevas funcionalidades o, como se realizará en este apartado, nueva información que estará a disposición de los usuarios. En concreto, se agregarán datos sobre nuevas enfermedades, para que el usuario pueda informarse de varias enfermedades y no solo de una, como pasa en el prototipo original del *bot*, que solo informa sobre el melanoma.

En esta fase del proyecto se ha decidido añadir una enfermedad minoritaria nueva, para que el *bot* sea capaz de informar sobre un total de dos enfermedades y, a partir de ahí, sería posible a nivel teórico la implementación de tantas enfermedades como fueran necesarias, dado que la estructura ya estaría formada. La enfermedad escogida para ser agregada es la Esclerosis Lateral Amiotrófica (ELA), que es una enfermedad minoritaria que se hizo bastante conocida hace unos años.

Para implementar las nuevas enfermedades, se utilizará, para almacenar la información y los hospitales, la misma estructura que utilizan los datos del melanoma; se guardará la información en diccionarios que contendrán un título y un contenido (en un archivo JSON). Dicha información se almacenará en la misma base de datos de Redis que se utiliza para los datos del melanoma. En el siguiente subapartado se analizará en profundidad la implementación de la enfermedad minoritaria elegida, explicando su naturaleza y las modificaciones necesarias a realizar en el programa para poder añadirla.



3.2.3.1. Esclerosis Lateral Amiotrófica (ELA)

La ELA, o Esclerosis Lateral Amiotrófica (ALS en inglés), es una enfermedad neurodegenerativa progresiva que afecta a las células nerviosas de la médula espinal y el cerebro. Afecta principalmente al sistema motor del cuerpo humano, provocando la degeneración progresiva de las neuronas motoras y, por tanto, afectando a los músculos de todo el cuerpo, llegando a provocar en las personas que la padecen la perdida de capacidades tan básicas como son hablar, comer o incluso respirar. Se ha elegido implementar esta enfermedad por la buena visibilidad que se le dio hace unos años, considerando así que será una buena forma de poder hacer que la herramienta del *chatbot* se haga conocer con más facilidad en un futuro.

El primer paso de todos para implementar esta nueva enfermedad es la búsqueda de información veraz, objetivo importante del proyecto. A diferencia del melanoma, para la ELA se añadirá una cantidad de información más reducida, ya que el verdadero objetivo es implementar la enfermedad y comprobar que todo funcione correctamente, dejando el sistema listo para poder modificar o añadir tantas enfermedades se quiera en un futuro. La información utilizada para que el *chatbot* informe de la ELA se ha extraído de la página web de una asociación que trata concretamente esta enfermedad (13).

Para añadir la información extraída sobre la ELA a la base de datos del *bot*, se ha creado primeramente un archivo JSON para almacenarlo todo, siguiendo la misma estructura de los datos del melanoma, ya que es necesaria para que el *bot* pueda gestionarlo correctamente. Dicha estructura está formada por una lista donde se almacenan, en un orden concreto, diversos diccionarios que contienen la información de la enfermedad. Cada diccionario es un bloque de datos, y está formado por el título, el contenido y la fuente de la información. También se crea un archivo con la información de varios hospitales donde tratan esta enfermedad, utilizando el mismo método que se usó con el melanoma; se crea un archivo de texto con los nombre de los hospitales y se pasa por la función “*generate_hospitals()*” (A4.4), que genera automáticamente un archivo JSON con la información concreta del hospital.

Una vez creados los archivos JSON, se procede a subir la información a la base de datos de Redis, tal como se hizo en el apartado 3.2.1.2 con el melanoma; en este caso concreto, se utilizarán los nombres ‘sclerosis_data’ y ‘sclerosis_hosp’ para las claves que contendrán los datos empaquetados de la enfermedad y los hospitales, respectivamente, manteniendo así un formato concreto para los nombres de todas las claves.

3.2.3.2. Modificaciones necesarias

Para que sea posible seleccionar la nueva enfermedad introducida en el *chatbot*, es necesario añadir el nombre de esta en la lista de enfermedades que se crea al principio del archivo principal ([A4.1](#)), para que, cuando el *bot* diga de especificar la enfermedad, pueda comparar el texto del mensaje enviado con los nombres de las enfermedades guardados en la lista; así, si se escribe un mensaje en el cual aparezca la palabra ‘*sclerosis*’, se seleccionará la ELA como la enfermedad sobre la cual se informará a partir de ese momento, cargando los datos almacenados en Redis. Una vez implementado todo se realiza una prueba para comprobar que el funcionamiento sea correcto, pero se encuentra un error a la hora de seleccionar la enfermedad.

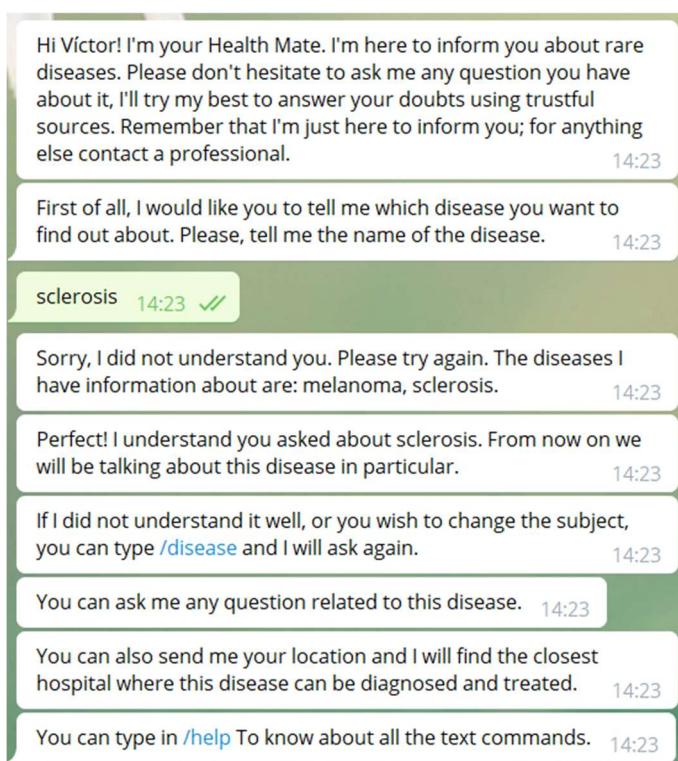


Figura 13. Captura de la conversación con el *bot*, donde se puede observar cómo aparece el mensaje equivocado a pesar de haber seleccionado correctamente la ELA como enfermedad para informar.

Al seleccionar la ELA mandando un mensaje con la palabra ‘*sclerosis*’, el *bot* establece correctamente la enfermedad, pero también envía, antes de eso, el mensaje programado para aparecer cuando no se ha especificado una enfermedad disponible para informar (Figura 13). Este comportamiento es erróneo, y sucede porque, a pesar de que el prototipo inicial se diseñó teniendo en cuenta la escalabilidad a la hora de añadir enfermedades, dentro del código del programa no se estableció una estructura correcta que considerase el hecho de existir más de una enfermedad al mismo tiempo, motivo por el cual aparece este error. La solución, pero, es muy sencilla, ya que solo se debe añadir un

condicional en la parte del código que gestiona ese mensaje, dentro de la función “*onMessage()*” ([A4.1](#)), para que solo se envía en el caso de haber recorrido toda la lista de enfermedades disponibles; así, si el programa ve que no aparece ‘melanoma’ en el texto, no enviará ese mensaje, pero sí que lo hará si ha visto que ‘*sclerosis*’, tampoco está, ya que se habrá recorrido la lista de enfermedades entera y no falta ninguna por comprobar. Con este simple cambio, por muchas enfermedades que se pudieran añadir en un futuro, nunca volverá a saltar ese mensaje a no ser que de verdad no se haya especificado ninguna de las enfermedades disponibles para el *bot*.

Otra modificación que se ha creído necesaria ha sido la transformación de las siglas ‘ALS’, que aparecen en la información de la enfermedad, al nombre completo (‘Amyotrophic Lateral Sclerosis’). Este tratamiento también se ha aplicado a las consultas de los usuarios. Esto se ha ideado con la finalidad de evitar que los modelos de cálculo de STS encuentren resultados erróneos al intentar comparar las siglas con el nombre completo, ya que no es seguro que puedan identificar el significado de las siglas, sobre todo si no se tiene en cuenta el contexto o si no se han entrenado con esa información concreta. Para realizar esta conversión se han utilizado expresiones regulares, concretamente se ha usado el módulo de Python ‘re’ (14), que sirve para realizar operaciones con expresiones regulares; simplemente se ha substituido de todas las partes del texto las siglas ‘ALS’ por su nombre completo.

4. Conclusiones

En este proyecto se ha llevado a cabo la implementación de un *chatbot* de Telegram para dar apoyo a los pacientes, y familiares de estos, que sufren enfermedades raras, gracias a la capacidad de informar utilizando datos veraces y contrastados. El origen del inicio de este proyecto se origina en el proyecto europeo Share4Rare, que lucha por dar visibilidad a este tipo de enfermedades y por impulsar sus investigaciones. Otro estudiante de la UPC revivió la herramienta que se diseñó en un *hackathon* organizado para la plataforma Share4Rare, basando su proyecto de final de grado en ella. De la misma forma, este proyecto ha seguido con el desarrollo del *chatbot* que diseñó el estudiante, con el fin de alcanzar una versión mejorada de esta herramienta, para que esté a un paso más de convertirse en una ayuda real.

Los objetivos específicos que se plantearon al inicio del proyecto fueron los de hacer una puesta a punto del prototipo del *chatbot* con el que se partió para solucionar errores; implementar un sistema de gestión de datos que utilizará servidores en línea; mejorar el sistema de búsqueda de respuestas; aumentar la base de datos con nuevas enfermedades. Todos estos objetivos han sido cumplidos en mayor o menor medida.

Se ha conseguido un *chatbot* capaz de informar de un total de dos enfermedades raras, el melanoma y la ELA, con una estructura preparada para añadir en un futuro todas las enfermedades que se quiera sin complicaciones; tiene capacidad de trabajo con servidores en línea, que permite almacenar toda la información necesaria sin necesidad de depender de archivos locales, además de haberse implementado un mejor sistema de lectura de los datos por parte del *bot*; también se ha potenciado la capacidad de búsqueda de resultados frente a consultas de los usuarios, mejorando y aplicando nuevos modelos de procesamiento del lenguaje, además de haber creado un sistema que permite sugerir múltiples respuestas al mismo tiempo, disminuyendo así la probabilidad de error frente a una consulta.

En definitiva, se ha logrado conseguir una versión un poco más estable del *chatbot* pero que no pretende ser nada más que un primer paso para la futura actualización de esta herramienta, con el fin de que algún día se pueda implementar en la plataforma Share4Rare. El resultado obtenido, después de haber solventado todos los obstáculos encontrados en el camino del proyecto, es satisfactorio, pero destacando aún así el gran margen de mejora que existe. Es por eso que se redactará, a modo de recomendaciones, los diferentes aspectos que pueden ser mejorados y nuevas ideas que podrían ser implementadas en el *chatbot* en un futuro.

4.1. Recomendaciones

Tras haber concluido el desarrollo del proyecto como tal, se ha decidido buscar vulnerabilidades en el funcionamiento del programa, con la finalidad de facilitar el futuro avance de esta herramienta; gracias a estos apuntes, será más sencillo localizar los diversos problemas y mejoras que se indicarán a continuación.

Tras ejecutar varias veces el programa y buscando vulnerabilidades y puntos débiles del *bot*, se han detectado las siguientes anomalías:

- Si no se detiene la ejecución del programa y se vuelve a iniciar, por mucho que se reinicie el chat de Telegram (comando '/start'), el *bot* creerá que ya se le ha hablado por primera vez, así que no seguirá la línea de conversación inicial.
- Si el *bot* no detecta el idioma a la primera, no se puede cambiar a no ser que se reinicie o se utilice el comando '/start'. Sería conveniente implementar un comando específico para el cambio de idioma. Además, con el comando '/start', cuando se quiere cambiar de idioma lo que hace es buscar directamente información de la enfermedad, así que se debería mostrar algún mensaje conforme se ha modificado correctamente.
- La estructura de árbol de decisiones no se aplica a las primeras dos interacciones con el chat (establecer idioma y enfermedad), por lo que, aun que intentes, por ejemplo, mandar un saludo, el *bot* no te lo devolverá.
- Si se envía la ubicación o si se utiliza el comando '/AllHospitals' antes de establecer la enfermedad, saltan errores que pueden detener la ejecución del programa. Se debería contemplar estas situaciones para evitarlo.
- Si se pregunta por dos enfermedades al mismo tiempo, el *bot* te informa solo de la primera que hay en la lista del programa. Se debería contemplar esto realizando alguna modificación en la conversación del *bot*.

5. Análisis del impacto ambiental

Dada la naturaleza de proyecto, el cálculo del impacto ambiental del mismo se convierte en una tarea difícil. Al estar diseñando e implementando un sistema de software, no aparecen aspectos tan influyentes como la fabricación o transporte de piezas y diferentes elementos de hardware, lo que tendría un claro impacto medible en el medioambiente; todo el material empleado, en especial el ordenador portátil que se ha utilizado, que ha sido el hardware utilizado por excelencia en este proyecto, tiene un claro origen industrial, donde la huella de carbono sería posible de calcular, asociando el impacto medioambiental de las diferentes fases por las que ha pasado el producto. A pesar de que todo esto es posible a nivel teórico, no se tienen los datos necesarios para realizar ningún cálculo y, por tanto, aún que se tiene en cuenta que esto es una realidad, no se puede mostrar ningún valor ni análisis más exhaustivo. De igual manera pasa con los consumos de los servidores de varios servicios y aplicaciones utilizados, como, por ejemplo, Telegram, que claramente provocan una huella de carbono, que a falta de datos es imposible de calcular.

Lo único que se puede medir de forma aproximada es la huella de carbono asociada al consumo de energía que presenta el ordenador portátil, y será el indicador utilizado para calcular el impacto ambiental asociado al proyecto. Teniendo en cuenta que la potencia media de un ordenador es de 220W (15), y que se considera que en el transcurso del proyecto se han invertido 600 horas (asociadas a un TFG de 24 créditos ECTS), se obtiene un consumo de energía de 132000Wh, o 132kWh. Utilizando una calculadora de emisiones de gases en función del consumo eléctrico (16), se obtienen unas emisiones de 92,5kg de CO₂.

De esta manera, se puede concluir que el impacto ambiental causado por este proyecto es mayor a unas emisiones de **92,5kg de CO₂**, ya que hay otros factores que, a pesar de no poder calcularse, incrementarían este valor.

6. Análisis Económico

En este apartado se realiza un estudio del coste económico asociado a este proyecto, teniendo en cuenta los diferentes tipos de coste, entre ellos el asociado a los recursos humanos y el asociado a los recursos materiales. Primeramente, se realiza un desglose de estos costes, para posteriormente hacer una evaluación global.

6.1. Coste de los recursos humanos

Para el coste asociado a los recursos humanos se tendrá en cuenta las horas de dedicación establecidas para un trabajo de final de grado de 24 créditos ECTS, que son 600 horas.

Para el precio asociado a estas horas se considerará que un desarrollador de software cobra de media unos 18€ la hora; se ha hecho una estimación superficial en función de los salarios encontrados en un portal de ofertas de empleo (17).

Tabla 3. Coste asociado a los recursos humanos (coste de personal).

Concepto	Precio (€/h)	Horas de trabajo (h)	Coste final (€)
Coste de personal	18	600	10800

6.2. Coste de los recursos materiales

Para el coste asociado a los recursos materiales se ha tenido en cuenta el material utilizado para realizar el proyecto (ordenador portátil), las licencias de los programas usados (Office), el precio de la tarifa de internet y el gasto por consumo eléctrico. Para este último, se utilizará el valor de consumo calculado en el apartado anterior, y se tendrá en cuenta que el precio asociado al consumo de electricidad es de 0,14 céntimos por kilovatio; esto resulta en un total de 18,48€.

También se considerará que, al haber optado a la matrícula adicional para alargar el plazo de entrega del proyecto, se añade un cuatrimestre más de dedicación al proyecto y, por tanto, se considerará que la duración del proyecto ha sido de un total de 8 meses

Tabla 4 Coste asociado a los recursos materiales.

Concepto	Precio (€/unidad)	Amortización (meses)	Duración del proyecto (meses)	Coste final (€)
Ordenador portátil	690,00	48	8	115
Licencia Office 365	69,00	12	8	46
Tarifa Internet	28,95	X	8	231,6
Consumo eléctrico	18,48	X	8	18,48
Total				411,08

6.3. Coste total asociado al proyecto

El coste total asociado al proyecto no es más que la suma de los costes anteriores (costes de recursos humanos y costes de material).

El coste total asociado a este proyecto es de **11211,08€**

Bibliografía

1. Tallada Castañé, M. Estudi per la creació d'un chatbot per a malalties minoritàries MEMÒRIA. En: [en línea]. 2020. [consulta: 19 abril 2021]. Disponible en: <https://upcommons.upc.edu/bitstream/handle/2117/329958/tfg-mar-al-tallada.pdf?sequence=1&isAllowed=y>.
2. Enfermedades Raras: preguntas frecuentes | FEDER. En: [en línea]. [consulta: 29 abril 2021]. Disponible en: <https://enfermedades-raras.org/index.php/enfermedades-raras/preguntas-frecuentes>.
3. El proyecto | Share4Rare. En: [en línea]. [consulta: 29 abril 2021]. Disponible en: <https://www.share4rare.org/es/el-proyecto-share4rare>.
4. Bakker, S. Historias del hackathon de Share4Rare: ayudando a pacientes y familias a afrontar las enfermedades raras | Share4Rare. En: [en línea]. [consulta: 29 abril 2021]. Disponible en: <https://www.share4rare.org/es/news/historias-del-hackathon-de-share4rare-ayudando-pacientes-y-familias-afrontar-las-enfermedades>.
5. Welcome to GeoPy's documentation! — GeoPy 2.1.0 documentation. En: [en línea]. [consulta: 19 abril 2021]. Disponible en: <https://geopy.readthedocs.io/en/stable/>.
6. Welcome to redis-py's documentation! — redis-py 2.10.5 documentation. En: [en línea]. [consulta: 19 abril 2021]. Disponible en: <https://redis-py.readthedocs.io/en/stable/>.
7. Pickle: Python object serialization — Python 3.9.4 documentation. En: [en línea]. [consulta: 19 abril 2021]. Disponible en: <https://docs.python.org/3/library/pickle.html>.
8. json — JSON encoder and decoder — Python 3.9.4 documentation. En: [en línea]. [consulta: 1 mayo 2021]. Disponible en: <https://docs.python.org/3/library/json.html>.
9. Cristian, I. Memory use and speed of JSON parsers | ionel's codelog. En: [en línea]. [consulta: 19 abril 2021]. Disponible en: <https://blog.ionelmc.ro/2015/11/22/memory-use-and-speed-of-json-parsers/#id14>.
10. Web Oficial. Linguistic Features · spaCy Usage Documentation. En: [en línea]. [consulta: 19 abril 2021]. Disponible en: <https://spacy.io/usage/linguistic-features#vectors-similarity>.
11. semantic-text-similarity · PyPI. En: [en línea]. [consulta: 2 abril 2021]. Disponible en: <https://pypi.org/project/semantic-text-similarity/>.
12. Telegram Web. Bots: An introduction for developers. En: [en línea]. [consulta: 19 abril 2021]. Disponible en: <https://core.telegram.org/bots>.
13. Understanding ALS | The ALS Association. En: [en línea]. [consulta: 29 abril 2021]. Disponible en: <https://www.als.org/understanding-als>.
14. Regular expression operations — Python 3.9.4 documentation. En: [en línea]. [consulta: 19 abril

- 2021]. Disponible en: <https://docs.python.org/3/library/re.html>.
15. Los ordenadores también emiten CO₂ | Ecoembes. En: [en línea]. [consulta: 2 mayo 2021]. Disponible en: <https://www.ecoembes.com/es/planeta-recicla/blog/los-ordenadores-tambien-emiten-co2>.
16. Greenhouse Gas Equivalencies Calculator | Energy and the Environment | US EPA. En: [en línea]. [consulta: 2 mayo 2021]. Disponible en: <https://www.epa.gov/energy/greenhouse-gas-equivalencies-calculator>.
17. Search jobs | Page Personnel. En: [en línea]. [consulta: 2 mayo 2021]. Disponible en: <https://www.pagepersonnel.es/jobs>.
18. Petit, J. Lliçons: Bots de Telegram. En: [en línea]. [consulta: 19 abril 2021]. Disponible en: <https://lliçons.jutge.org/python/telegram.html>.
19. Escudero, G. y Petit, J. Python 3: telegram bots. En: [en línea]. [consulta: 19 abril 2021]. Disponible en: <https://gebakx.github.io/Python3/chatbots.html#1>.
20. SEPAR. *Convivir con la esclerosis lateral amiotrófica* [en línea]. AlaOeste C. Editorial Respira, 2018. ISBN 9788494777196. Disponible en: <https://www.separ.es/node/1376>.
21. Escudero, G. y Turmo, J. Introduction to Human Language Technologies: Framework. En: [en línea]. [consulta: 19 abril 2021]. Disponible en: <https://gebakx.github.io/ihlt/s1/#1>.
22. Welcome to Python Telegram Bot's documentation! — Python Telegram Bot 13.4.1 documentation. En: [en línea]. [consulta: 19 abril 2021]. Disponible en: <https://python-telegram-bot.readthedocs.io/en/stable/index.html>.
23. Web Oficial. Orphanet - El portal sobre enfermedades raras y medicamentos huérfanos. En: [en línea]. [consulta: 19 abril 2021]. Disponible en: <https://www.orpha.net/consor/cgi-bin/index.php>.
24. Web Oficial. Redis. En: [en línea]. [consulta: 19 abril 2021]. Disponible en: <https://redis.io/>.
25. Solomon, B. How to Use Redis With Python – Real Python. En: [en línea]. [consulta: 19 abril 2021]. Disponible en: <https://realpython.com/python-redis/>.
26. Tyagi, N. What is spaCy in Natural Language Processing (NLP)? | Analytics Steps. En: [en línea]. [consulta: 19 abril 2021]. Disponible en: <https://www.analyticssteps.com/blogs/what-spacy-natural-language-processing-nlp>.
27. Ferreira, D. What are Sentence Embeddings and why are they useful? | Talkdesk Engineering. En: [en línea]. [consulta: 19 abril 2021]. Disponible en: <https://engineering.talkdesk.com/what-are-sentence-embeddings-and-why-are-they-useful-53ed370b3f35>.
28. Raj, S. *Sumit Raj - Building Chatbots with Python_ Using Natural Language Processing and Machine Learning-Apress (2019)*. 2018. ISBN 9781484240953.
29. Semantic textual similarity | NLP-progress. En: [en línea]. [consulta: 19 abril 2021]. Disponible en: http://nlpprogress.com/english/semantic_textual_similarity.html.

30. Share4Rare. En: [en línea]. [consulta: 28 abril 2021]. Disponible en: <https://www.share4rare.org/es>.
31. Òmada Interactiva. Sare4Rare | Comunidad de pacientes. Plataforma de investigación en enfermedades raras. En: [en línea]. [consulta: 28 abril 2021]. Disponible en: <https://www.omada.es/es/#portfolio>.
32. Investigació, innovació, ciència i compromís | Recerca Sant Joan de Déu. En: [en línea]. [consulta: 28 abril 2021]. Disponible en: <https://www.sjdrecerca.org/ca/>.
33. Fernández, Y. API: qué es y para qué sirve. En: [en línea]. [consulta: 29 abril 2021]. Disponible en: <https://www.xataka.com/basics/api-que-sirve>.
34. Markdown - La guía definitiva en español. En: [en línea]. [consulta: 29 abril 2021]. Disponible en: <https://markdown.es/>.
35. González, A. ¿Qué es Machine Learning? – Cleverdata. En: [en línea]. [consulta: 29 abril 2021]. Disponible en: <https://cleverdata.io/que-es-machine-learning-big-data/>.
36. Arrabales, R. Deep Learning: qué es y por qué va a ser una tecnología clave en el futuro de la inteligencia artificial. En: [en línea]. [consulta: 29 abril 2021]. Disponible en: <https://www.xataka.com/robotica-e-ia/deep-learning-que-es-y-por-que-va-a-ser-una-tecnologia-clave-en-el-futuro-de-la-inteligencia-artificial>.
37. Merayo, P. ¿Qué es la correlación estadística y cómo interpretarla? | Máxima Formación. En: [en línea]. [consulta: 29 abril 2021]. Disponible en: <https://www.maximaformacion.es/blog-dat/que-es-la-correlacion-estadistica-y-como-interpretarla/>.
38. Ferreira, D. What are Word Embeddings and why are they useful? | by Diogo Ferreira | Talkdesk Engineering. En: [en línea]. [consulta: 29 abril 2021]. Disponible en: <https://engineering.talkdesk.com/what-are-word-embeddings-and-why-are-they-useful-a45f49edf7ab>.
39. Qué son las redes neuronales y sus funciones | ATRIA Innovation. En: [en línea]. [consulta: 29 abril 2021]. Disponible en: <https://www.atriainnovation.com/que-son-las-redes-neuronales-y-sus-funciones/>.
40. Lucca, R. ¿Qué es BERT y cómo funciona? - DXmedia. En: [en línea]. [consulta: 29 abril 2021]. Disponible en: <https://dxmedia.net/algoritmo-bert-google/>.
41. JSON. En: [en línea]. [consulta: 29 abril 2021]. Disponible en: <https://www.json.org/json-es.html>.
42. Pipeline - GlosarioIT: Glosario Informático. En: [en línea]. [consulta: 29 abril 2021]. Disponible en: <https://www.glosarioit.com/Pipeline>.
43. Ruelas, U. ¿Qué es la serialización o marshalling? En: [en línea]. [consulta: 29 abril 2021]. Disponible en: <https://codingornot.com/que-es-la-serializacion-o-marshalling>.
44. Requena, B. Rango intercuartílico. En: [en línea]. [consulta: 29 abril 2021]. Disponible en: <https://www.universoformulas.com/estadistica/descriptiva/rango-intercuartilico/>.

45. Lozano, J.J. Programación orientada a objetos (POO) en Python. En: [en línea]. [consulta: 1 mayo 2021]. Disponible en: <https://j2logo.com/python/tutorial/programacion-orientada-a-objetos/>.
46. ¿Qué es la geocodificación?—Ayuda | ArcGIS for Desktop. En: [en línea]. [consulta: 1 mayo 2021]. Disponible en: <https://desktop.arcgis.com/es/arcmap/10.3/guide-books/geocoding/what-is-geocoding.htm>.
47. UTF-8: codificación para una comunicación digital global - IONOS. En: [en línea]. [consulta: 1 mayo 2021]. Disponible en: <https://www.ionos.es/digitalguide/paginas-web/creacion-de-paginas-web/utf-8-codificacion-para-una-comunicacion-digital-global/>.
- (18)(19)(20)(21)(22)(23)(24)(25)(26)(27)(28)(29)(30)(31)(32)(33)(34)(35)(36)(37)(38)(39)(40)(41)(42)(43)(44)(45)(46)(47)

Anexo A: contenido del archivo adjunto

En este anexo se encuentra una explicación de todo el contenido presente en el archivo adjunto a la memoria, esto es, todos los códigos de Python que confeccionan el bot y otros archivos utilizados en el transcurso del proyecto. Se ha separado este anexo en diversos apartados donde se explica el contenido de cada carpeta y sus respectivos archivos, focalizando sobre todo en los archivos Python que permiten ejecutar el chatbot y sus diversas funcionalidades.

A1. Carpeta “Hospitales”

En la carpeta de hospitales, tal como el nombre indica, es donde se almacenan los archivos de texto con los nombres de los hospitales que tratan una enfermedad en concreto. El nombre de estos archivos tiene un formato concreto (“hospitals+’nombre_enfermedad’.txt”), dado que se utilizan para generar los datos de ubicación de los hospitales que posteriormente el bot utilizará en algunas de sus funciones.

A2. Carpeta “ProcessedData”

Aquí se almacenan los archivos, en formato JSON, que contienen los datos sobre las enfermedades que trata el chatbot. El nombre de los archivos tiene un formato concreto tanto para los hospitales (“hospitals_information_’nombre_enfermedad’.json”) como para la información de las enfermedades (“main_data_’nombre_enfermedad’.json”), ya que se utilizan para subir los datos a la base de datos de Redis.

A3. Carpeta “Test STS”

En esta carpeta se encuentra el programa de Python utilizado para realizar el cálculo de correlación de los diferentes modelos capaces de calcular la similitud semántica de dos textos, que ha servido para justificar y demostrar la mejora que suponen los modelos basados en Sentence Embedding respecto a los que utilizan Word Embedding. Este cálculo se ha realizado mediante un test que utiliza unos datos concretos, llamados ‘test-gold’, incluidos también en la carpeta. Este programa es una modificación realizada de una primera versión entregada por el tutor de este proyecto, Gerard Escudero, con el fin de adaptarlo a las necesidades específicas del proyecto.

A4. Carpeta “Códigos”

Aquí se encuentran todos los archivos Python que contienen los códigos que hacen funcionar al bot, junto a todas sus funcionalidades, así como los programas utilizados para el procesamiento previo de los datos. Para dejar claro el contenido de cada archivo presente en esta carpeta, se ha realizado una separación en diferentes apartados, donde se comentan los diferentes bloques de código y funciones que se han utilizado para cada uno de ellos.

A4.1. Archivo “bot.py”

Este archivo constituye el programa como tal que estructura el chatbot. En él se realiza la conexión y configuración del bot de Telegram, esto es, se establecen todos los comandos y funcionalidades que tendrá presente y que permitirán la interacción deseada con el usuario. Es además el archivo principal que se debe ejecutar, desde el cual se abre la base de datos de Redis, se cargan los demás archivos, librerías y módulos necesarios para su correcto funcionamiento y presenta variables globales necesarias para la gestión del bot.

La función “onMessage(bot, update, user_data)”, la única que se ha modificado de este archivo respecto al prototipo, es el núcleo del bot, ya que se ejecuta cada vez que un usuario manda un mensaje, siendo la más extensa de todas y conteniendo las características más importantes. Se podría considerar que está dividida en bloques, dado que la función utiliza una estructura de condicionales muy marcada. Se mantienen muchos elementos del prototipo inicial, incluida la estructura, pero se han modificado ciertos elementos y añadido bloques condicionales necesarios para las nuevas implementaciones que se han realizado en los apartados X y X. El resto de funciones no se han visto afectadas por los nuevos cambios y, por tanto, no ha sido necesario modificarlas.

A4.2. Archivo “utils.py”

El archivo “utils” contiene las diferentes funciones utilizadas como herramientas dentro del archivo principal (“bot”), a modo de módulo externo. Principalmente son funciones con código muy extenso o que no son dependientes directamente para el bot, sino para sus funcionalidades, como, por ejemplo, la carga de datos de enfermedades o el cálculo de la similitud semántica de dos textos. Gracias a esta manera de gestionar las funciones se ahorra mucho código en el archivo principal, aumentando así la comprensión y simplificación de este. Este es el archivo que se ha visto más modificado a causa de las nuevas características implementadas en el bot. Algunas de las funciones que aparecen no se utilizan en ningún momento, de igual manera que pasaba en el prototipo, pero se ha decidido no eliminarlas dado que podrían tener un futuro uso.

La función “read_book(key, r)”, utilizada anteriormente para cargar la información de las enfermedades y los hospitales desde archivos en formato JSON, ahora funciona de la misma manera, pero utilizando los datos almacenados en una base de datos de Redis. Este cambio corresponde con lo planteado e implementado en el apartado X.

El segundo cambio más importante corresponde al bloque de funciones involucradas en el cálculo de la similitud semántica entre dos textos. La función “process_message(question, data, multiple, model)”, utilizada dentro del archivo principal “bot.py”, ahora contiene nuevos parámetros que no presentaba el prototipo, permitiendo así escoger el modelo de STS y si se desea que el bot siempre de a elegir tres respuestas ante la consulta de un usuario. El parámetro del modelo se envía a través de la función “compute_scores_from_index(question, data, model)”, que acaba llegando a la función “get_score(question, data, model)”, donde se han cambiado más cosas respecto al prototipo inicial. Todos estos cambios se ven con mucho más detalle en el apartado X.

A4.3. Archivo “preprocessing.py”

Aquí se encuentra el código utilizado para el procesamiento de los datos tanto de enfermedades como de hospitales, esto es, se utiliza para subir todos los datos disponibles en la carpeta “ProcessedData” a la base de datos de Redis, utilizando la estructura diseñada en la segunda propuesta del apartado X. Para añadir información de nuevas enfermedades se debería ampliar el código de la misma manera que se ha realizado al añadir los datos actuales. La única función utilizada en este archivo es “read_json(file)”, que sirve para leer los datos previamente almacenados en formato JSON; esta era la antigua función que utilizaba el prototipo inicial para cargar los datos de las enfermedades.

A4.4. Archivo “CoordinatesFinder.py”

En este archivo se encuentra una única función, “generate_hospitals(disease)”, utilizada para generar la información de hospitales que tratan las distintas enfermedades sobre las que informa el bot. A partir de los nombres de los hospitales almacenados en los archivos de textos guardados en la carpeta “Hospitales”, se genera un archivo JSON con información más detallada de la localización de dichos hospitales, que se almacena en la carpeta “ProcessedData”. Este código se ha modificado, respecto al prototipo inicial, en la gestión de la localización de los archivos manipulados en el proceso, pero la funcionalidad del mismo no ha variado.

A4.5. Archivo “translate.py”

Este archivo se utiliza simplemente como un módulo de traducción, gracias a la librería que permite utilizar el traductor de Google; presenta dos funciones, “tr2english” y “tr2other”, que, como su propio nombre indica, sirven para traducir un texto al inglés o un texto en inglés al idioma que se indique, respectivamente. Para traducir al inglés, es necesario pasar por la función los datos del usuario de Telegram para extraer de ahí el idioma que utiliza; en caso de no estar indicado el idioma en esos datos, se utiliza una herramienta de detección de idioma que analiza el texto que se quiere traducir.

Respecto al prototipo inicial, solo se ha variado ligeramente la estructura de las funciones, pero no se han realizado modificaciones que varíen el comportamiento de las mismas. Estas modificaciones se realizaron con el fin de solucionar un error que empezó a suceder a mitad del transcurso del proyecto, que provocaba el funcionamiento errático del módulo de traducción; este error era debido al uso excesivo que se le dio al módulo al ejecutar pruebas sobre el bot, que desencadenó en un bloqueo de la IP del ordenador por parte de Google, que bloqueaba casi completamente el uso del traductor. Esto no se pudo solucionar con las modificaciones introducidas, ya que no es un problema que se pueda solucionar con código, pero no supone ningún impedimento a la hora de implementarlo en el bot, ya que al utilizar el programa en otro dispositivo el funcionamiento es correcto.

Anexo B: Plantilla de evaluación

En este anexo se presentan capturas de pantallas de conversaciones con el chatbot como método de evaluación de su correcto funcionamiento en caso de introducir cambios en su estructura interna. Parte de esta plantilla (apartado X) se ha utilizado en este mismo proyecto con el fin de asegurar que las variaciones e implementaciones de nuevas características no provocaran un comportamiento erróneo; el resto de esta se incluye como futura medida de corrección ante posibles alteraciones no deseadas del funcionamiento del bot.

Esta plantilla no tiene en cuenta las posibles mejoras directas en la capacidad de cálculo de la STS, o los posibles cambios de información sobre las enfermedades disponibles, por tanto, se debe tener en cuenta estos factores a la hora de considerar que la variación del comportamiento es errónea, pudiendo ser en estos casos más acertado.

Para conseguir una visualización más clara, se divide este anexo en dos apartados, que corresponden a la primera versión de la plantilla elaborada previamente al desarrollo de las modificaciones y nuevas características, y a las posibles nuevas líneas de conversación posteriores a la finalización del proyecto.



B1. Plantilla inicial (general + melanoma)

Esta es la plantilla de evaluación elaborada al inicio del desarrollo del proyecto que se ha utilizado como guía del buen comportamiento del bot. Se recopila una primera conversación en inglés que abarca todas las posibles interacciones con el usuario, siguiendo una estructura lineal (las capturas aparecen en el orden real de la conversación). Además, se recopila el inicio de la conversación en dos idiomas más (español y catalán) para la verificación de la funcionalidad de traducción.

- **Conversación en inglés.**

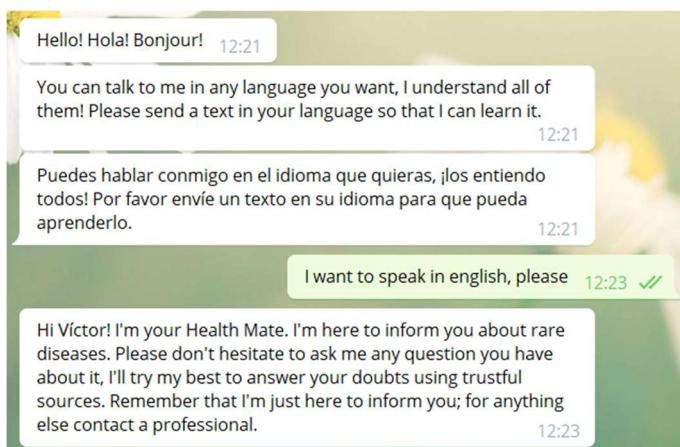


Figura B1. Conversación inicial (comando 'start'). Se establece el inglés al escribir un texto en este idioma.

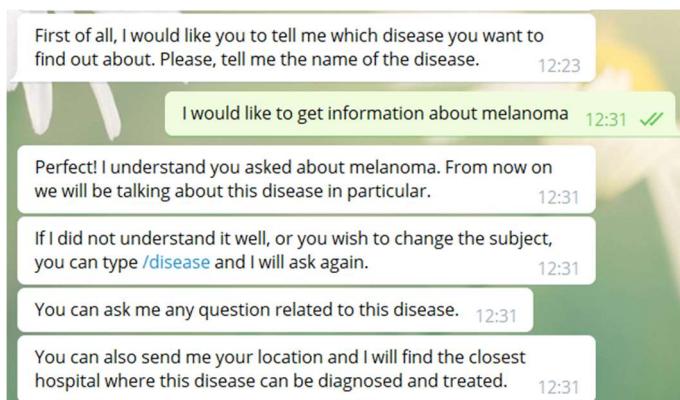


Figura B2. Captura de la conversación con el bot en inglés. Se establece la enfermedad (melanoma) sobre la que informará el bot.

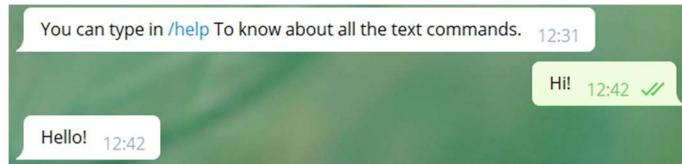


Figura B3. Captura de la conversación con el bot en inglés. Se le saluda y él devuelve el saludo.

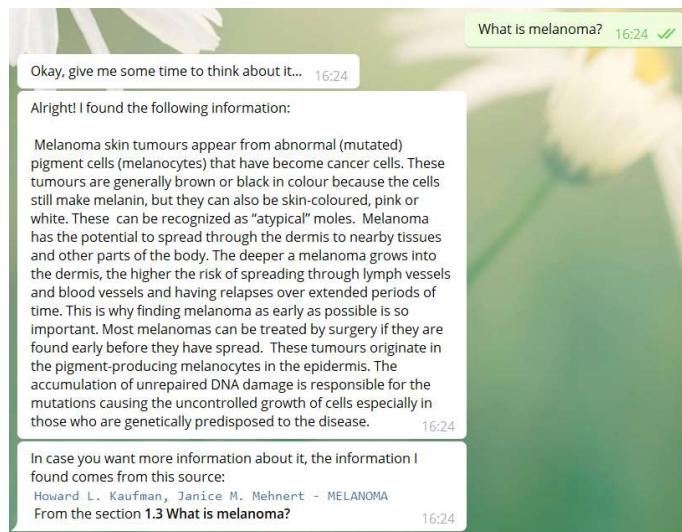


Figura B4. Captura de la conversación con el bot en inglés. Se pregunta qué es el melanoma.

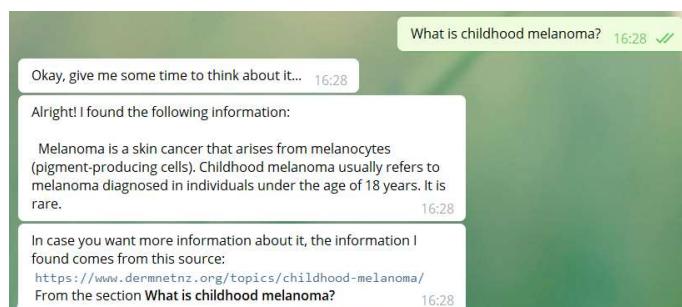


Figura B5. Captura de la conversación con el bot en inglés. Se pregunta qué es el melanoma infantil.

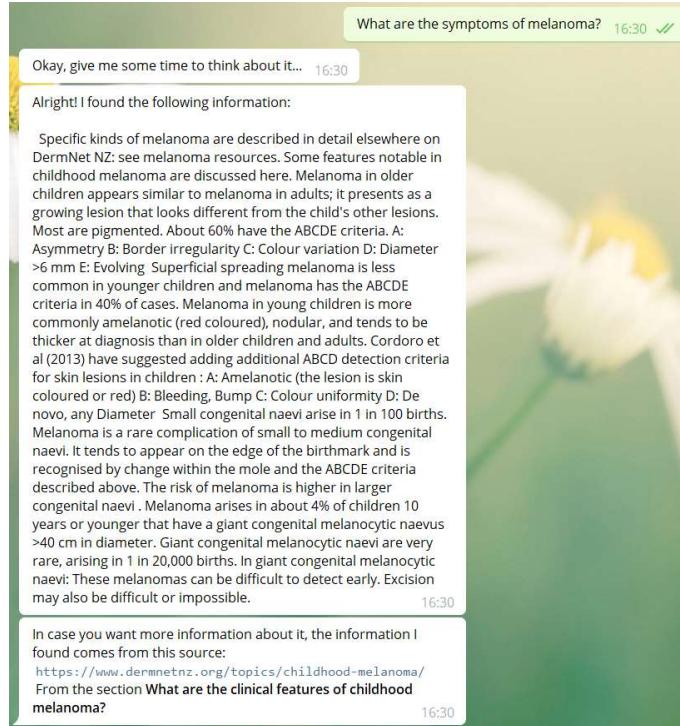


Figura B6. Captura de la conversación con el bot en inglés. Se hace una consulta preguntando por los síntomas del melanoma.

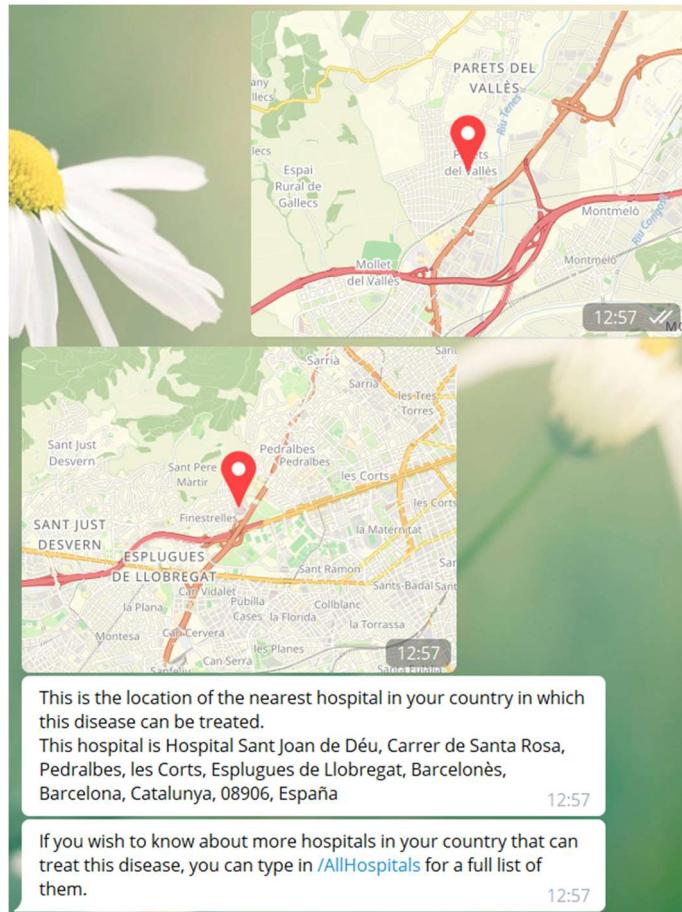


Figura B7. Captura de la conversación con el bot en inglés. Se envía la ubicación para recibir la dirección del hospital más cercano.



Figura B8. Captura de la conversación con el bot en inglés. Se utiliza el comando 'AllHospitals' para recibir la dirección de todos los hospitales (a los que el bot tenga acceso) donde tratan el melanoma.

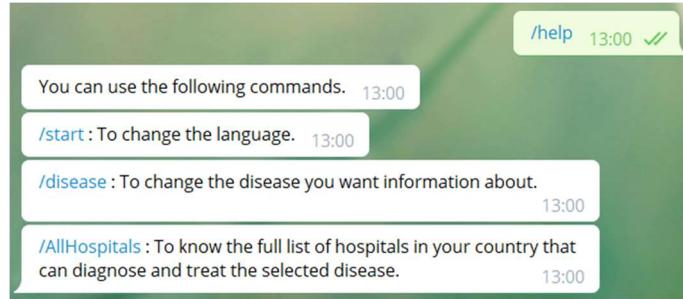


Figura B9. Captura de la conversación con el bot en inglés. Se utiliza el comando ‘*help*’ para obtener todos los comandos disponibles.

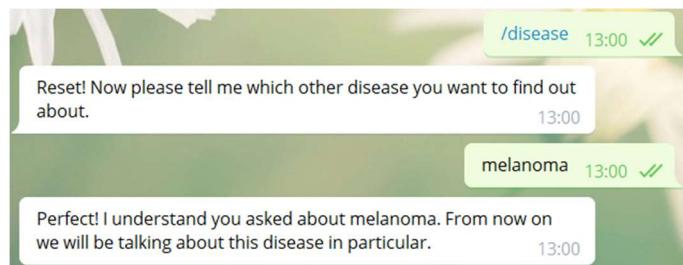


Figura B10. Captura de la conversación con el bot en inglés. Se utiliza el comando ‘*disease*’ para volver a seleccionar la enfermedad de la que se quiere obtener información.



Figura B11. Captura de la conversación en inglés, donde se agradece la ayuda al bot.

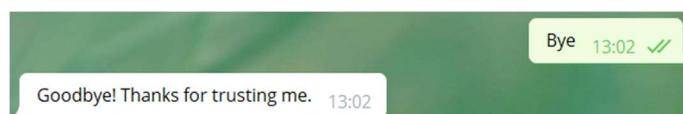


Figura B12. Captura de la conversación en inglés, donde se despide al bot.

- Conversación en español

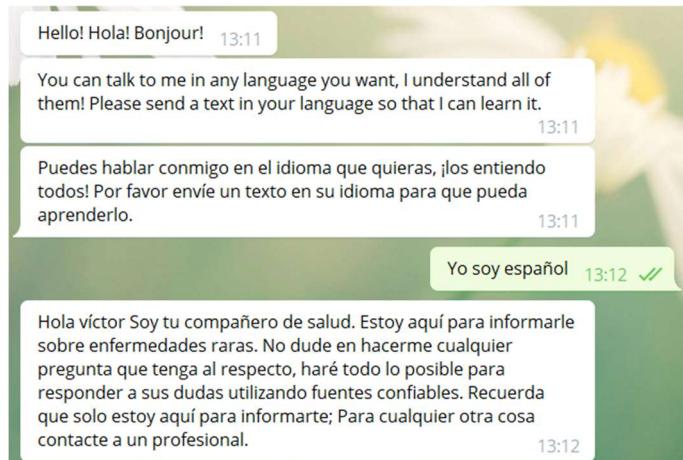


Figura B13. Captura de la conversación con el bot. Se establece el idioma en español al escribir un texto en este idioma.

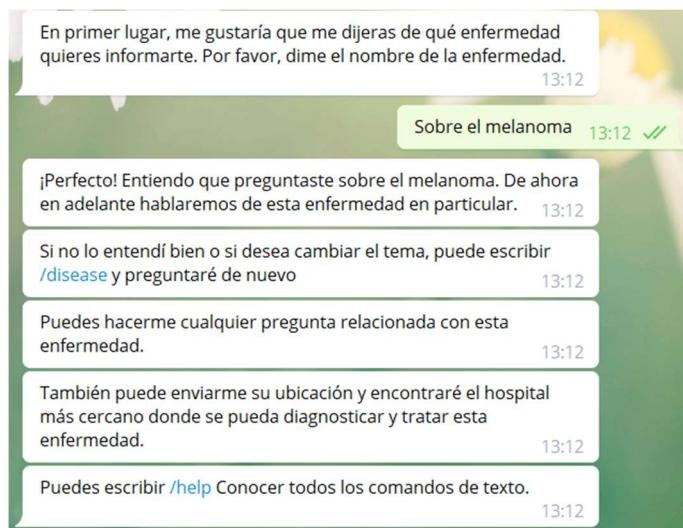


Figura B14. Captura de la conversación con el bot en español. Se establece la enfermedad (melanoma) sobre la que informará el bot.



Figura B15. Captura de la conversación con el bot en español. Se pregunta qué es el melanoma.

• Conversación en catalán



Figura B16. Captura de la conversación con el bot. Se establece el idioma en catalán al escribir un texto en este idioma.

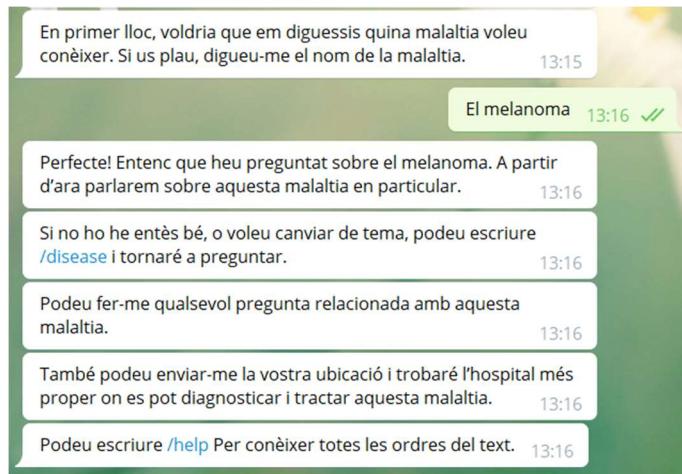


Figura B17. Captura de la conversación con el bot en catalán. Se establece la enfermedad (melanoma) sobre la que informará el bot.



Figura B18. Captura de la conversación con el bot en catalán. Se pregunta qué es el melanoma.

B2. Líneas de conversación adicionales (ELA + múltiple respuesta)

En este apartado se recopilan las nuevas líneas de conversación que han surgido a causa de la implementación de nuevas características. Principalmente se han listado consultas sobre la ELA y ejemplos de respuestas múltiples por parte del bot. Solamente se han añadido líneas de conversación en inglés, ya que, al no haber sido modificada la traducción de los textos, sigue funcionando correctamente con los nuevos casos. En estos casos de la plantilla se ha utilizado en todo momento el modelo “ClinicalBertSimilarity” para el cálculo de STS.

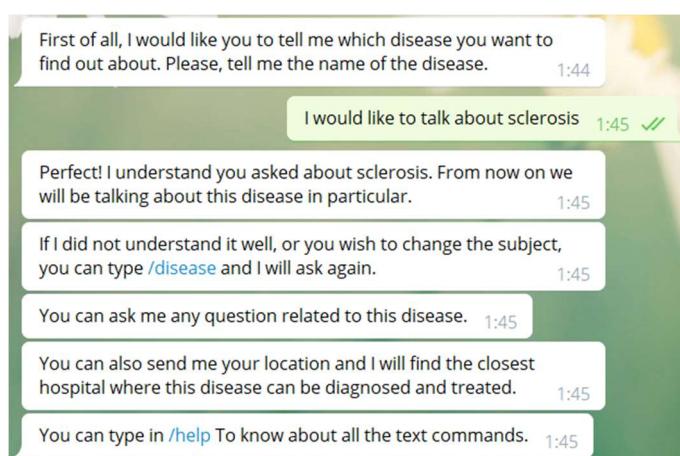


Figura B19. Captura de la conversación con el bot. Se establece la enfermedad (ELA) sobre la que informará el bot.

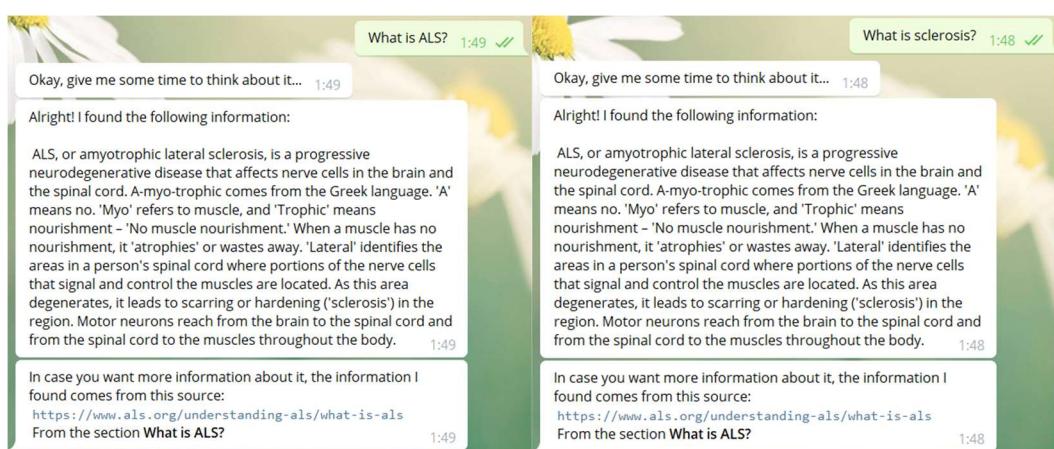


Figura B20. Captura de la conversación con el bot. Se pregunta qué es la ELA/sclerosis.

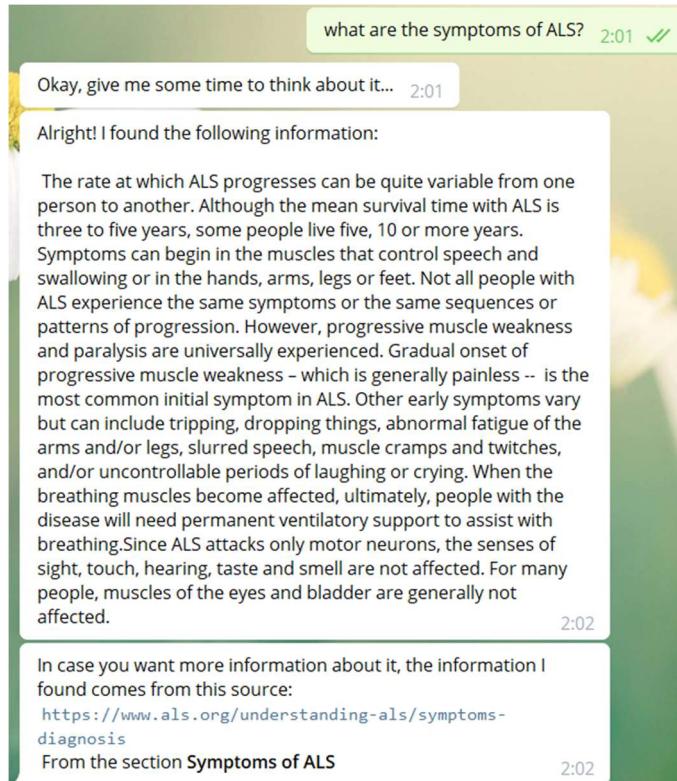


Figura B21. Captura de la conversación con el bot. Se pregunta cuales son los síntomas de la ELA.

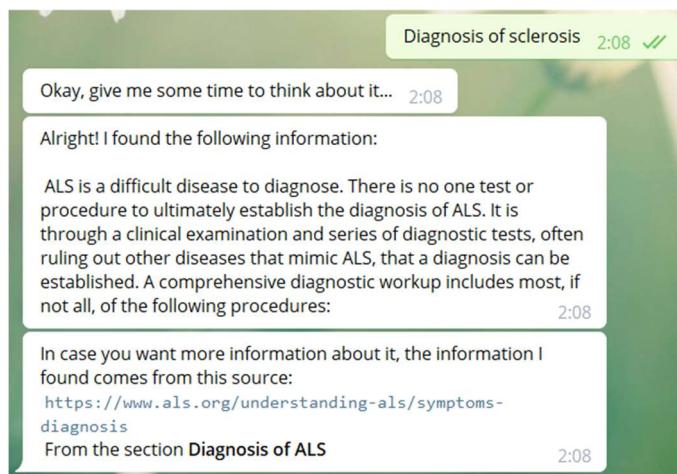


Figura B22. Captura de la conversación con el bot. Se pregunta sobre la diagnosis de la ELA.

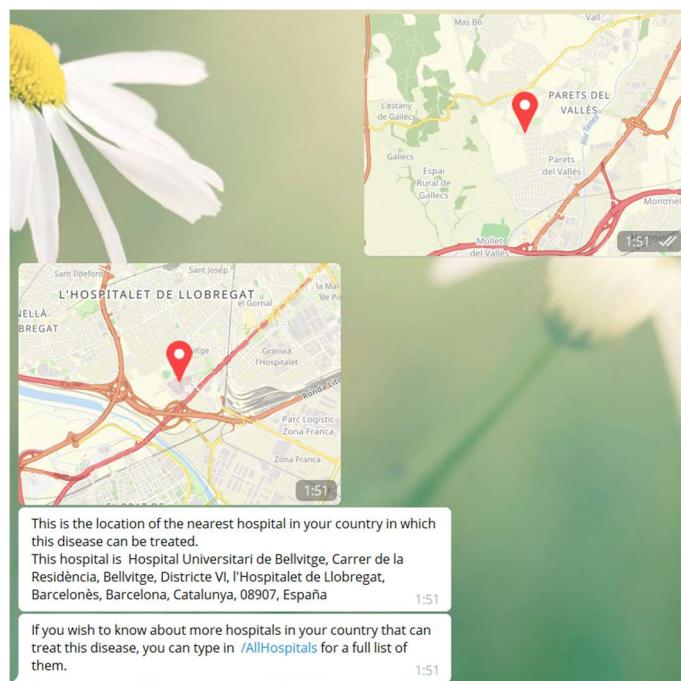


Figura B23. Captura de la conversación con el bot. Se envía la ubicación para recibir la dirección del hospital más cercano.



Figura B24. Captura de la conversación con el bot. Se utiliza el comando 'AllHospitals' para recibir la dirección de todos los hospitales (a los que el bot tenga acceso) donde tratan el melanoma.

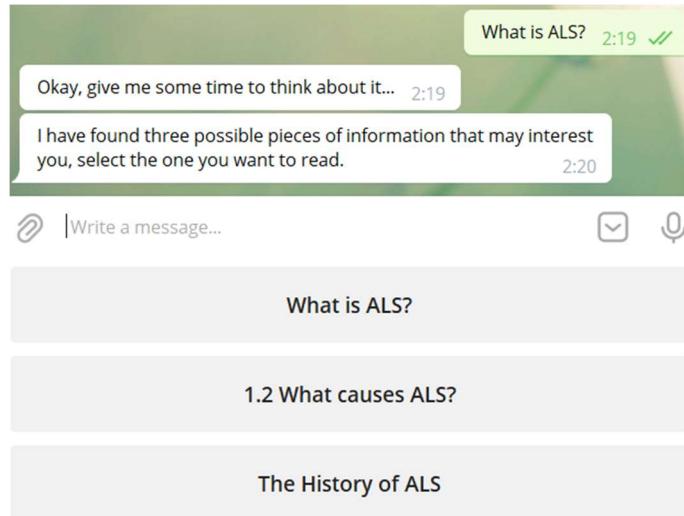


Figura B25. Captura de la conversación con el bot. Se pregunta qué es la ELA y el bot devuelve las tres entradas más parecidas que ha encontrado.

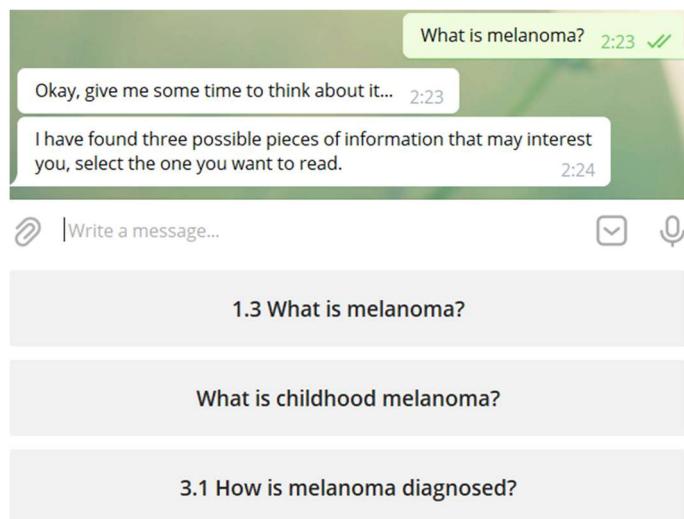


Figura B26. Captura de la conversación con el bot. Se pregunta qué es el melanoma y el bot devuelve las tres entradas más parecidas que ha encontrado.