Universidad Internacional de La Rioja (UNIR)

ESIT

Máster Universitario en Inteligencia Artificial

Agente conversacional para registrar información de pacientes con diabetes mellitus tipo 1

Trabajo Fin de Máster

Presentado por: Paules Sánchez, Carlos

Director/a: Rodríguez León, Ciro

Ciudad: Barcelona
Fecha: 2 de febrero de 2022

Resumen

En este proyecto se ha creado un agente conversacional usable y funcional empleando Dialogflow, que utiliza técnicas de procesamiento del lenguaje natural para mantener conversaciones diarias con usuarios diabéticos tipo 1. La información sobre diabetes detectada se almacena en la base de datos Firebase. Se ha seguido una metodología iterativa e incremental para el diseño, implementación y testeo del agente con cuatro usuarios. El agente almacena información sobre niveles de glucosa, dosis de insulina, ejercicio físico, comida y situaciones causantes de estrés. También se ha creado una metodología para implementar respuestas dinámicas que finalmente no se ha podido aplicar por limitaciones de Dialogflow. La puntuación System Usability Scale del agente ha sido 85,5 demostrando que es una herramienta usable. La mitad de respuestas generadas por el modelo son adecuadas en la conversación excepto en mensajes de glucosa e insulina debido al reducido tamaño de la base de datos de entrenamiento.

Palabras Clave: agente conversacional, diabetes mellitus tipo 1, procesamiento del lenguaje natural, creación de bases de datos, Dialogflow.

Abstract

The focus of this project was to create a usable and functional chatbot using Dialogflow to maintain daily conversations with type 1 diabetic users. Whenever diabetes information was detected, it should be stored in Firebase database. An iterative and incremental methodology was used to design, develop and test the chatbot with four users. The agent created was able to store information about glucose levels, insulin doses, physical exercise, food ingested and situations that cause stress to the user. A methodology was also created to implement dynamic responses that couldn't be applied to the final version due to Dialogflow limitations. The System Usability Scale score obtained by the agent was 85,5 showing it is a usable tool. Half of the responses generated by the model are correct for the conversation except for glucose and insulin message's response, due to the reduced size of the training database.

Keywords: chatbot, type 1 diabetes mellitus, natural language generation, database creation, Dialogflow.

Índice de contenidos

Índice de tablas

Índice de figuras

1 Introducción

La Diabetes Mellitus (DM) es una enfermedad crónica que afecta a 463 millones de adultos entre 20 y 79 años (un 9,3% de la población mundial) (International Diabetes Federation, 2019), cuya prevalencia sigue aumentando cada año en niños y adultos. Existen varios tipos de diabetes de entre los cuales destacan dos: la Diabetes Mellitus tipo 1 (DM1), que está causada por un déficit en la producción natural de insulina, y la Diabetes Mellitus tipo 2 (DM2), en la cual el organismo no es capaz de utilizar la insulina eficazmente. El tratamiento para la DM1 es la inyección de insulina diaria, cuya dosis depende principalmente de la actividad física y de la ingesta de carbohidratos a lo largo del día. Por otro lado, la DM2 depende del grado de fallo de absorción de insulina del organismo del enfermo y se puede tratar de varias formas. En casos más leves, se suele tratar con un cambio del estilo de vida del enfermo a uno más saludable, incorporando más actividad física y controlando la dieta. En los más avanzados se suele requerir también la ingesta de pastillas que facilitan esta absorción o la inyección de insulina (International Diabetes Federation, 2019).

La DM tiene riesgos a corto plazo y a largo plazo. Una mala gestión de esta enfermedad puede conllevar complicaciones a largo plazo en los pacientes como la pérdida de visión (retinopatía diabética), un deterioro del sistema nervioso, problemas en los pies y enfermedades cardíacas y cardiovasculares, entre otras. La mayoría de estas complicaciones son inicialmente causadas por un mal funcionamiento de los vasos sanguíneos, debido a un mal control de la glucosa en sangre. En el día a día, un enfermo de DM1 puede llegar a dos extremos que son perjudiciales: la hiperglucemia y la hipoglucemia. La hiperglucemia causa deshidratación y un malestar general, y la hipoglucemia causa un mal funcionamiento del cerebro debido a una falta de glucosa. Una hipoglucemia puede tener complicaciones como el desmayo o, en un caso extremo, la muerte. Los enfermos de DM2 que no se inyectan insulina únicamente tienen riesgo de hiperglucemia. Por tanto, un buen control del nivel de glucosa en sangre es esencial para un diabético.

También existen otros factores que influyen en un buen control de la DM. Una dieta controlada y saludable ayuda a predecir mejor los valores de glucosa. La actividad física regular facilita la gestión de la DM, reduciendo la cantidad de insulina por cada dosis en pacientes con DM1 y reduciendo la cantidad de tratamiento necesario para enfermos con DM2. Algunas enfermedades, estrés y periodos menstruales pueden conllevar un aumento del nivel de glucosa en sangre debido a que afectan al equilibrio hormonal del cuerpo (American Diabetes Association, 2018). Podemos concluir que un tema esencial en una buena gestión de la

enfermedad es una concienciación por parte de los enfermos de DM y una consideración y control adecuado de estos elementos.

En la actualidad, la inteligencia artificial (IA) está innovando multitud de campos y disciplinas distintas. La mayoría de los sistemas operativos de cualquier ordenador o dispositivo móvil tienen integrados un asistente conversacional que se utiliza para cualquier finalidad. Estos asistentes se diseñan con técnicas avanzadas de procesamiento del lenguaje natural (PLN), con las cuales es posible que los agentes comprendan el lenguaje y sean capaces de responder de una forma adecuada. Existen multitud de asistentes conversacionales para lenguaje escrito (*chatbots*) y lenguaje oral, que mantienen conversaciones más generales o se especializan en temas concretos. Algunos ejemplos conocidos de asistentes conversacionales de voz son Siri de la empresa Apple (Apple, s.f.), Alexa de Amazon (Amazon, 2019) o Cortana de Microsoft (Microsoft, Cortana - Your personal productivity assistant., 2019).

Un asistente conversacional puede aportar mucho valor en la gestión de la DM debido a que ésta es una enfermedad que depende de un control regular. Por medio de una conversación o un sistema de preguntas y respuestas, un asistente es capaz de recopilar información del diabético y utilizarla para predecir eventos futuros de glucosa fuera de rango o realizar recordatorios. De hecho, estos asistentes aplicados al soporte de enfermos de diabetes han demostrado ser una gran ventaja para el tratamiento de la DM2 en personas de edad avanzada (Magyar, y otros, 2019).

Además del PLN, también existen otras ramas de la IA que han demostrado un aporte beneficioso en la gestión de la DM. Varios grupos de investigación han creado modelos de aprendizaje automático capaces de predecir valores de glucosa futuros en función de valores actuales o pasados y muchos otros factores. En el trabajo de Alfian et al. se muestra una tabla con algunos de los modelos creados hasta el momento, los factores que se han tenido en cuenta y la base de datos utilizada (Alfian, y otros, 2020). Algunos de estos modelos trabajan con bases de datos muy grandes que se han obtenido manualmente, y otros utilizan bases de datos que se han obtenido de registros de hospitales.

1.1 Motivación

La recopilación y almacenamiento de datos útiles sobre el control de la DM de un paciente combinado con el uso de herramientas que empleen técnicas de IA puede aportar gran valor en la gestión de la enfermedad. Actualmente existen aplicaciones que emplean técnicas de PLN para ayudar a sus usuarios a llevar un mejor control de la DM. Estas aplicaciones utilizan

la información extraída y almacenada sobre la DM del usuario para generar recordatorios, recomendaciones o consejos relacionados con la enfermedad. Sin embargo, ninguna de estas aplicaciones está focalizada en la creación de bases de datos con información sobre DM.

De esta forma, se detecta una escasez de sistemas que se centren en la adquisición y registro de datos relevantes para la diabetes utilizando asistentes conversacionales. Los datos que se utilizan actualmente para aplicar IA a diabetes suelen ser creados por cada grupo de investigación y están orientados a la finalidad de cada agente o proyecto, que normalmente es predecir eventos futuros, generar recordatorios y sugerir buenas prácticas para la gestión de la diabetes. Si se pudiera obtener una base de datos estandarizada con todo tipo de información, se podría usar con distintas finalidades y de forma más global.

Además, una conversación diaria con un agente permitiría adquirir una cantidad mucho mayor de datos más precisos que los que se le da al doctor en visitas mensuales. Estos datos pueden ayudar a descubrir información relevante para el control de la diabetes que no se haya tenido en cuenta hasta el momento. Al no tener en cuenta el seguimiento, el asistente se centrará totalmente en el registro de datos.

1.2 Planteamiento del trabajo

En este proyecto se pretende crear una herramienta accesible e interactiva para el almacenamiento de datos de pacientes con DM1. Estos datos se podrán utilizar en el futuro con cualquier finalidad, como por ejemplo entrenar modelos de IA que permitan predecir con más precisión niveles de glucosa futuros. La herramienta ha de registrar cualquier tipo de dato que sea útil para la predicción del nivel de glucosa en sangre futura, desde valores de glucosa o los factores anteriores hasta cualquier otra información nueva que pueda afectar.

Esta herramienta consistirá en un asistente conversacional que mantendrá una conversación de texto escrito con la persona diabética. A través de esta conversación, el asistente usará técnicas de PLN para comprender toda la información, seleccionar qué partes hay que almacenar en la base de datos y generar respuestas a los mensajes del usuario de forma autónoma.

1.3 Estructura de la memoria

En el contexto y estado del arte se aporta información en detalle sobre la DM y sus tipos, el PLN, las técnicas que se emplean en esta disciplina y una de sus aplicaciones más conocidas,

los agentes conversacionales. También se estudian los diversos proyectos de investigación centrados en el desarrollo de agentes conversacionales para usuarios con DM y se comenta el resultado de la búsqueda de bases de datos de DM para entrenar este tipo de agentes. Finalmente se extraen las conclusiones del estudio del contexto y se determinan las herramientas a utilizar. En el siguiente apartado se definen el objetivo principal y los específicos de este proyecto, además de explicar en detalle la metodología de desarrollo de software iterativa e incremental que se ha usado. En el cuarto apartado se lleva a cabo una identificación de requisitos globales o funcionalidades esperadas del sistema, y requisitos específicos de las herramientas usadas para poder empezar con la creación del agente. El siguiente apartado aporta una explicación detallada del proceso de desarrollo del agente conversacional utilizando las herramientas concretadas en el segundo apartado. También da detalles sobre el desarrollo del modelo de generación de lenguaje natural (GLN) y explica las limitaciones halladas al final de la implementación. En el sexto apartado se explican las evaluaciones del agente y del modelo que se han llevado a cabo para comprobar que los objetivos establecidos se han cumplido. Finalmente se comentan las conclusiones obtenidas tras la finalización del proyecto, las posibles formas de mejorar la herramienta creada y la aportación del mismo al estado actual de la investigación en este campo.

2 Contexto y estado del arte

2.1 Diabetes Mellitus

La DM es una enfermedad crónica que causa un aumento de la concentración de glucosa en sangre debido a un déficit en la producción de la insulina o a un uso incorrecto de ésta por parte del cuerpo. La insulina es una hormona que se produce en el páncreas y cuya función es retirar la glucosa del torrente sanguíneo e introducirla en las células del cuerpo para que allí se metabolice y se transforme en energía. Si se controlan los niveles de glucosa en sangre dentro de unos límites se puede llevar una vida longeva sin complicaciones. Sin embargo, mantener un nivel de glucosa en sangre alto habitualmente puede dañar algunos órganos y causar complicaciones en el futuro (International Diabetes Federation, 2019). Estas complicaciones suelen ser enfermedades cardiovasculares, problemas renales, degeneración del sistema nervioso y retinopatías (afectación ocular que en extremos puede llevar a la ceguera). La hiperglucemia afecta a los nervios distales de las extremidades, cosa que produce un entumecimiento en los pies y una pérdida de sensibilidad. Esto sumado a una disminución de la capacidad curativa del sistema sanguíneo conlleva a la creación de úlceras en el pie, que es lo que se conoce como “pie diabético”. Se diferencian principalmente dos tipos de DM, aunque existen también otras variaciones que no encajan totalmente en ninguno de los dos tipos.

2.1.1 Diabetes Mellitus Tipo 1

La DM1 es debida a una reacción autoinmune del cuerpo a las células beta del páncreas, que son las que producen insulina. Esto hace que la producción de insulina vaya disminuyendo hasta que se detiene por completo. Las causas de esta reacción autoinmune no se conocen con certeza y todavía se están estudiando, pero se sospecha que se debe a una predisposición genética combinada con otros factores ambientales. El tratamiento principal para la DM1 es la inyección de insulina diaria, que se combina con un estilo de vida saludable, dietas controladas y actividad física habitual para prevenir complicaciones futuras. La DM1 afecta principalmente a niños y jóvenes, y si estos viven en zonas con pocos recursos puede ser difícil mantener un estilo de vida saludable. Esto puede conllevar a la complicación más habitual para este tipo de diabetes que es la cetoacidosis diabética, en la cual se crean cuerpos cetónicos en las vías sanguíneas que son dañinos para el cuerpo. Esta y otras complicaciones tienen mayor probabilidad de aparecer cuanto más tarde en diagnosticarse la DM1 (American Diabetes Association, 2021).

Además de la cetoacidosis diabética, otro riesgo muy común en los enfermos de DM1 es la hipoglucemia. Esta sucede cuando no hay una concentración suficiente de glucosa en sangre debido a un desequilibrio entre la dosis de insulina, la actividad física y la ingesta de carbohidratos. El tratamiento recomendado para una hipoglucemia leve es la ingesta de 15g de carbohidratos de absorción rápida y pasados 20 minutos comprobar la glucosa de nuevo. Estos carbohidratos pueden ser en forma de azúcar, azúcar diluido en agua, 150mL de zumo o refresco, o 15mL de miel. Cuando este tratamiento es insuficiente para remontar la hipoglucemia, se considera una hipoglucemia grave y requiere la inyección de glucagón o de glucosa intravenosa. El glucagón es la hormona antagonista de la insulina, cuya función es liberar glucosa en el torrente sanguíneo (Yale, Paty, & A., 2018).

2.1.2 Diabetes Mellitus Tipo 2

En la DM2, el nivel de glucosa en sangre aumenta debido a que las células del cuerpo tienen más dificultad en absorber insulina. Esto produce un aumento en la producción de insulina, que a la larga puede afectar a las células pancreáticas, generando una cantidad insuficiente de insulina. Las causas de esta dificultad en la absorción de insulina tampoco se conocen totalmente, pero se sospecha que al igual que en la DM1 son una predisposición genética combinada con factores ambientales. La DM2 afecta más a gente mayor con problemas de obesidad y vida sedentaria, aunque también hay gran incidencia en niños con estilos de vida poco sanos. Por tanto, el tratamiento en este caso es principalmente la adopción de un estilo de vida saludable llevando una dieta controlada, realizando actividad física habitual, manteniendo un peso adecuado y no fumando. Cuando no se pueden controlar los niveles de glucosa en sangre con esto, se utilizan varios medicamentos ingeridos que aumentan la absorción de insulina. Como último recurso, la DM2 también se puede tratar con inyecciones de insulina en caso de que el resto de los tratamientos fallen. El diagnóstico de la DM2 es más difícil que el de la DM1 debido a que no siempre se muestran síntomas, cosa que puede llevar a complicaciones previas a la detección de la enfermedad (American Diabetes Association, 2021).

2.1.3 Otros tipos de Diabetes

Además de estos dos tipos de DM, existen otras afectaciones que también aumentan el nivel de glucosa en sangre. Una de ellas se conoce como prediabetes, que son la tolerancia anormal a la glucosa y la alteración de la glucosa en ayunas. En estos casos se modifica el nivel de glucosa en sangre, pero no lo suficiente como para ser considerado diabetes. Por tanto, estas afectaciones son pistas que pueden indicar que en un futuro se padecerá DM2. Por otro lado, existe la diabetes gestacional, que es la que padecen algunas mujeres durante

el embarazo. Este tipo de diabetes suele durar únicamente el periodo del embarazo y sucede con mayor probabilidad en mujeres con predisposición genética o que han ganado mucho peso. Padecer diabetes gestacional aumenta la probabilidad de padecer DM2 en un futuro de 3 a 6 años, y el bebé tiene riesgo también de padecer obesidad y DM2. Por último, existen otros factores que causan diabetes como la afectación de un único gen (diabetes monogénica), problemas pancreáticos, problemas hormonales con antagonistas de la insulina e infecciones víricas que afectan las células pancreáticas (que fabrican insulina) (International Diabetes Federation, 2019).

2.2 Procesamiento del lenguaje natural

2.2.1 Introducción e historia

Se conoce el PLN como la capacidad que tienen los sistemas computacionales de reconocer y entender el lenguaje humano, ya sea escrito o de voz. Por tanto, el PLN es una disciplina que combina la lingüística con la ciencia computacional, y se basa en la comprensión de la semántica (significados) y de la sintaxis (reglas) del lenguaje por parte de las computadoras. El PLN se utiliza con distintas finalidades, como por ejemplo la categorización de contenido de un texto, la detección de duplicaciones, el análisis de sentimiento, la transformación de lenguaje oral a escrito y viceversa, el resumen de textos y la traducción automática a distintos idiomas. El concepto de PLN apareció en los años 50, cuando se crearon los primeros sistemas de análisis de oraciones, traducciones y respuestas a preguntas. Estos sistemas utilizaban métodos basados en reglas (Ramesh, Ravishankaran, Joshi, & Chandrasekaran, 2017).

La creación de estas reglas la llevaban a cabo un grupo de expertos en el lenguaje y requerían grandes esfuerzos por su parte. Además, para ser capaz de comprender totalmente un lenguaje hace falta un conjunto muy grande de reglas, cosa que dificultaba el funcionamiento de estos sistemas. Posteriormente, en los años 90, se desarrollaron métodos de aprendizaje estadísticos que utilizaban datos etiquetados. Gracias al desarrollo de los modelos de aprendizaje automático estos métodos eran capaces de calcular probabilidades en función de los datos etiquetados de entrada. Esto conllevó una mejora en los sistemas de PLN con respecto a los métodos basados en reglas especialmente en traducción automática de textos. En 2012, la aplicación del aprendizaje profundo al PLN mostró una clara mejora con respecto a los métodos estadísticos, obteniendo mejoras en las distintas aplicaciones del PLN. En la actualidad sigue siendo el método más usado debido a los buenos resultados que

proporciona, pese a que únicamente funciona bien con un gran conjunto de datos etiquetados (Hien, Cuong, Nam, Nhung, & Thang, 2018).

2.2.2 Técnicas empleadas en el PLN

Para poder entrenar modelos de aprendizaje profundo en PLN se utiliza una técnica denominada *embedding*. Esta consiste en mapear palabras u oraciones a un espacio semántico, convirtiéndolas en vectores que serán más cercanas cuanto más cercanos sean sus significados. Estos vectores se introducen en modelos de aprendizaje profundo (se suelen usar las redes neuronales convolucionales), que devolverán una predicción en función del objetivo del PLN. En tareas de clasificación como el análisis de sentimiento, el modelo devolverá una valoración (que puede ser positiva o negativa o tener distintos grados) en función de una nueva oración, palabra o extracto de texto introducido. Esta técnica también se usa en tareas de etiquetado en las cuales se pretende hallar la categoría gramatical de las palabras o la sintaxis de una oración (sintagmas). Por último, estos modelos también se pueden utilizar para la generación de nuevas frases en agentes conversacionales, generación de textos o traducción automática (Zhou, Duan, Liu, & Shum, 2020).

El *embedding* de palabras puede hacerse teniendo o sin tener en cuenta el contexto de la palabra. Cuando no se tiene en cuenta el contexto se utilizan dos métodos: el *Bag-Of-Words* (BOW) (Mikolov, Chen, Corrado, & Dean, 2013) y los modelos de skip-gram. En el primero se predice una palabra en función de un grupo de palabras que la rodean, y está basado en el número de ocurrencias de estas palabras en el corpus. El corpus es el conjunto de textos de que se dispone y con el que trabajan los modelos. En el segundo se predice el grupo de palabras previo y posterior en función de una palabra central. En ambos métodos el *embedding* calculado es constante para cada palabra, y por eso se considera que no depende del contexto.

Cuando se tiene en cuenta el contexto de la palabra se pueden utilizar redes neuronales recurrentes (RNR), *embedding* basado en la atención propia y redes neuronales convolucionales (RNC) (Collobert & Weston, 2008). En el primer tipo se crea una RNR hacia delante que analiza cada palabra de la oración en orden y va creando para cada una un estado oculto que contiene información de dicha palabra con las anteriores. El segundo tipo también va analizando la oración en orden y almacenando información, pero en este caso se puede observar la interacción directa de cada palabra con cada una de las anteriores. En el último tipo se define una ventana de palabras alrededor de la que se analiza y se obtiene la información de cada palabra en relación con el resto de las palabras de la ventana. El cálculo del *embedding* de la frase completa se hará de forma distinta en los tres tipos. En las RNR se

mirará el estado oculto de la última palabra. En el *embedding* basado en atención propia se observará la información del último elemento de la oración, que viene después del signo de puntuación. En las RNC se pasará toda la información obtenida por una capa *MaxPooling*, que reducirá el tamaño quedándose con los valores máximos (Zhou, Duan, Liu, & Shum, 2020).

Para la traducción automática y conversaciones con *chatbots* se utilizan modelos secuencia a secuencia, en los cuales se genera una secuencia de palabras a partir de una secuencia de entrada. El modelo codificador-decodificador (Cho, y otros, 2014) utiliza el modelo RNR definido antes como un codificador, y el *embedding* de la última palabra lo lee otra RNR que funciona como decodificador. El decodificador va creando palabras de la nueva secuencia hasta llegar a un elemento considerado el final. Este modelo depende totalmente del último estado oculto de la RNR, y por tanto puede perder información de estados anteriores. Para solucionar esto, se utiliza otro modelo en el cual se ponderan todos los estados ocultos del codificador, y esto lo utiliza el decodificador para crear la nueva secuencia. En la Figura 1 se muestra el modelo de codificador-decodificador basado en la atención.

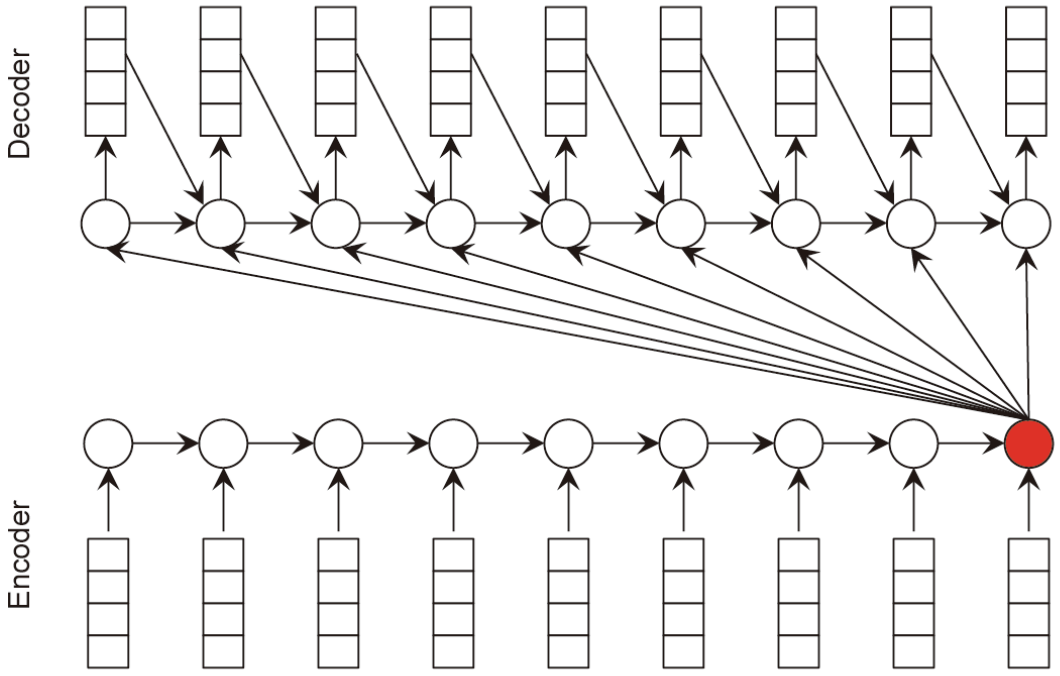

Figura 1: Modelo codificador-decodificador, donde el codificador funciona como un método de "embedding" de RNR. Los rectángulos representan los "embeddings" de cada palabra, y los círculos son los estados ocultos de la RNR (Zhou, Duan, Liu, & Shum, 2020).

2.2.3 Estudios centrados en PLN y DM

Existe una multitud de trabajos cuyo foco de estudio es la DM y el PLN. Se pueden diferenciar principalmente dos ramas de estudio. Por un lado, existen trabajos que se basan en la creación de una herramienta que emplea técnicas de PLN para aportar un beneficio para la DM. El concepto de la DM en el cual se focalizan estas herramientas es principalmente el diagnóstico de la enfermedad, y en menor medida la detección de hipoglucemias. Otros conceptos de la DM que también se investigan en un menor número de trabajos son la influencia del estilo de vida en la enfermedad o el rechazo de insulina de los pacientes. Por otro lado, hay trabajos que se centran en llevar a cabo una investigación clínica empleando técnicas de PLN (Turchin & Florez Builes, 2021).

2.3 Agentes conversacionales

2.3.1 Introducción e historia

Una aplicación del PLN muy usada en la actualidad son los agentes conversacionales, también denominados *chatbots*. Un *chatbot* es un programa que utiliza técnicas de IA (especialmente el PLN y análisis de sentimiento) para mantener conversaciones con usuarios humanos u otros *chatbots*. Estas conversaciones pueden ser de temáticas más generales o especializadas dependiendo del tipo de agente conversacional y de su objetivo. El análisis de sentimiento de un *chatbot* es imprescindible para mantener una conversación más adecuada y aumenta la aceptación por parte del interlocutor humano (Adamopoulou & Moussiades, 2020).

El inicio de los *chatbots* surge en los años 50 a raíz de lo que se denominó el test de Turing. Cuando un humano mantiene una conversación con un programa y no es capaz de identificar que no está conversando con un humano, se considera que el programa ha superado el test de Turing. El primer *chatbot* creado fue ELIZA (Weizenbaum, 1966) en 1966, que simulaba un psicoterapeuta y únicamente realizaba preguntas relacionadas con este tema. En 1972 se diseñó PARRY (Colby, Weber, & Hilf, 1971), un *chatbot* que simulaba un paciente con esquizofrenia. Este agente conversacional era capaz de mostrar emociones pese a que no tenía demasiada comprensión del lenguaje. En 1988 se inventó Jabberwacky, el primer agente conversacional que aplicaba IA al ser capaz de aprender de conversaciones previas. Posteriormente, en 1995, se creó un *chatbot* denominado ALICE (Wallace, 2009) que fue programado en un lenguaje diseñado específicamente para este fin, y era capaz de mantener

conversaciones más largas y de cualquier tema. En 2001 apareció SmarterChild (Molnár & Zoltán, 2018), el primer *chatbot* capaz de recopilar información de sus bases de datos acerca de cualquier tema e introducirlo en la conversación. En la actualidad existe una multitud de agentes conversacionales de todo tipo. Algunos asistentes muy conocidos son los de voz, que entienden los comandos de sonido y mantienen un diálogo con una voz digital. Ejemplos conocidos de estos asistentes son Siri de Apple (Apple, s.f.), Watson de IBM (IBM, Intelligent Virtual Agent - IBM Watson Assistant, 2020), el asistente de Google (Google, Google assistant, your own personal google., 2019), Cortana de Microsoft (Microsoft, Cortana - Your personal productivity assistant., 2019) o Alexa de Amazon (Amazon, 2019).

2.3.2 Técnicas de desarrollo de agentes conversacionales

Existen dos técnicas principales para desarrollar *chatbots*: *pattern matching* y aprendizaje automático. El *pattern matching* es un algoritmo basado en reglas, en el cual el agente contrasta el mensaje de entrada con las reglas establecidas en su base de datos, y devuelve la respuesta más adecuada de su conjunto de respuestas predefinidas. Por tanto, cuanto mayor sea la base de datos del *chatbot*, mayor capacidad de respuesta tendrá. Las limitaciones de esta técnica son que no es muy robusta frente a las faltas de ortografía de los inputs, que suelen no tener en cuenta el contexto (únicamente la última respuesta) y que no tienen la misma espontaneidad que una respuesta humana. Los tres lenguajes más comunes para la implementación de *chatbots* usando *pattern matching* son Artificial Intelligence Markup Language (Marietto, y otros, 2013), RiveScript (Petherbridge, 2009) y ChatScript (Wilcox & Wilcox, 2014).

Por otro lado, los *chatbots* que utilizan técnicas de aprendizaje automático usan PLN para extraer la información del mensaje de entrada y poder elaborar una respuesta. Por lo tanto, estos agentes sí que tienen en cuenta el contexto y pueden hacer respuestas más espontáneas, pero para ello necesitan entrenarse con conjuntos de datos muy grandes. El PLN en agentes conversacionales se divide en dos partes principales: la comprensión del lenguaje natural (McShane, 2017) y la GLN. En la comprensión del lenguaje natural, el agente busca clasificar la intención del mensaje y extraer la entidad teniendo en cuenta el contexto. La entidad de una palabra es la categoría a la que pertenece dicha palabra dentro de un conjunto de categorías predefinidas (lugar, persona, organización, tiempo, cantidad, etc.). Para la clasificación de intenciones se usan modelos clasificadores que categorizan el mensaje recibido dentro de un tema definido. La extracción de entidades también usa modelos que permiten que el agente reconozca la estructura del mensaje recibido. La GLN se realiza creando los *embeddings* de las palabras e introduciéndolos en las redes neuronales, que producen automáticamente la respuesta.

2.3.3 Estructura interna de un agente conversacional

Por lo general, los componentes de un *chatbot* siguen una arquitectura general, que se puede ver modificada dependiendo de la técnica de desarrollo del agente conversacional y de la función que desempeña. La Figura 2 muestra los cinco componentes principales de la arquitectura de un *chatbot*: la interfaz de usuario, el análisis de mensaje del usuario, el gestor de diálogo, el *backend* o base de datos y el generador de respuesta (Adamopoulou & Moussiades, 2020).

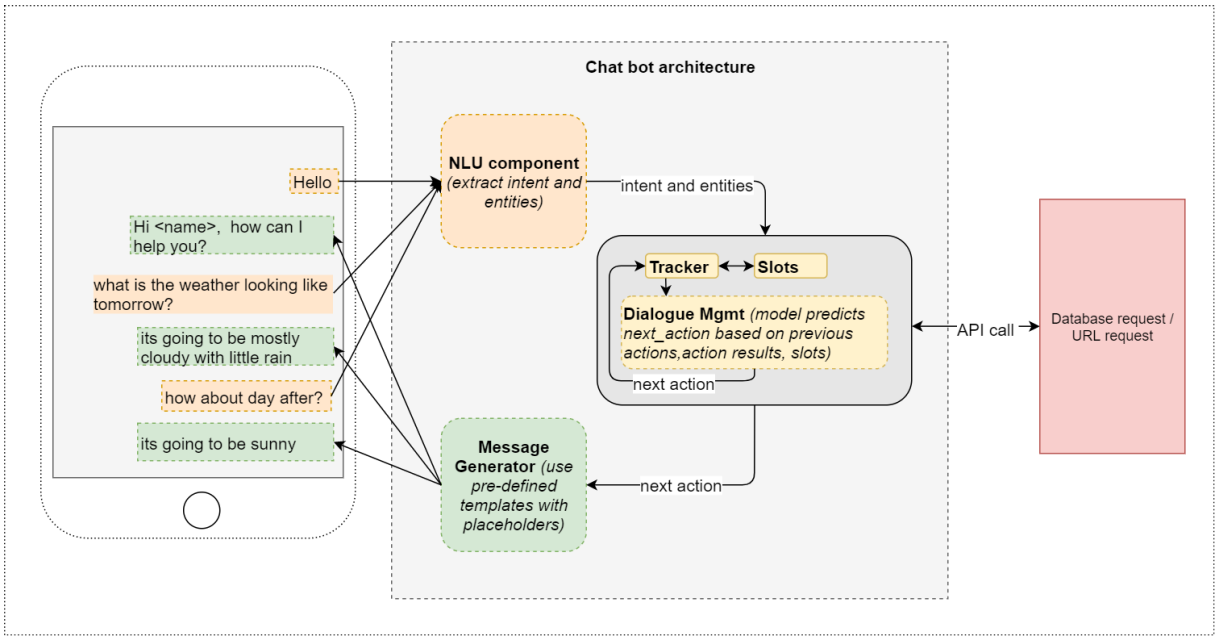

Figura 2: Arquitectura básica de un chatbot (Kompella, 2018)

La interfaz de usuario es la plataforma por la cual se realiza la conversación, es decir, por donde el usuario envía los mensajes (de texto o audio) al agente conversacional y recibe respuestas. Esta interfaz suele estar integrada en una aplicación focalizada en conversaciones, como puede ser Facebook, Slack, WhatsApp, Skype, WeChat, Viber o Telegram.

El componente de análisis de mensaje del usuario está formado por un módulo de comprensión de lenguaje natural. Este componente recibe el mensaje de la interfaz de usuario y extrae la intención del mensaje y las entidades que se encuentran en él. También contiene herramientas que facilitan estas tareas como un corrector ortográfico, un traductor automático (en caso de que el *chatbot* pueda comunicarse en varios idiomas) y un módulo de análisis de sentimiento que detecta si el mensaje es positivo o negativo (Chowdhury, 2003).

El componente de gestión de diálogo contiene información acerca del contexto de la conversación, y se ocupa de mantener la intención y entidades actualizadas. Este

componente contiene tres módulos para mantener la información del contexto (Kucherbaev, Bozzon, & Houben, 2018). El primero es la gestión de la ambigüedad, que sucede cuando el *chatbot* no es capaz de identificar la intención. Este devuelve respuestas para clarificar la intención, respuestas generales que puedan abarcar la intención no detectada correctamente o comienza una nueva conversación. Por tanto, este componente es el que decide cómo continuar la conversación en función de la intención y entidades recibidas.

El componente de *backend* contiene las bases de datos con la información aprendida. En el caso de agentes conversacionales basados en reglas, este componente contiene una base de conocimiento con un conjunto de respuestas posibles escritas a mano. La base de conocimiento también puede ser generada automáticamente o estar basada en ontologías. Así, este componente es utilizado por el gestor de diálogo para obtener la información necesaria para cumplir la intención del mensaje del usuario (Khanna, y otros, 2015).

Por último, el componente de generación de respuestas utiliza modelos basados en reglas, modelos de recuperación o modelos generativos para crear un mensaje y devolverlo por la interfaz de usuario. En los modelos basados en reglas, el generador de respuestas recibe los valores correspondientes a la respuesta que provienen de la base de conocimiento (Ramesh, Ravishankaran, Joshi, & Chandrasekaran, 2017). Los modelos de recuperación eligen la respuesta más adecuada explorando en recursos disponibles mediante APIs (Hien, Cuong, Nam, Nhung, & Thang, 2018). Por último, el modelo generativo utiliza GLN, que consiste en redes neuronales recurrentes que van creando una oración a partir de la información analizada (Singh, Darbari, Bhattacharjee, & Verma, 2016). Entrenar estas redes recurrentes para obtener buenos resultados requiere una cantidad masiva de datos, cosa que dificulta la producción de este tipo de modelos (Kim, Lee, Kim, Lee, & Kim, 2018).

2.3.4 Plataformas de desarrollo de agentes conversacionales

Existen principalmente dos tipos de plataformas para desarrollar agentes conversacionales: las comerciales y las de fuente abierta (Nayyar, 2019). Las plataformas de fuente abierta tienen todo el código de programación abierto, y por tanto se puede tener un control completo de la implementación del *chatbot*. Ejemplos de plataformas de fuente abierta son Botkit (Brown, Nasirzada, & Benson, 2021), RASA (Rasa, 2021), Pandorabots (Pandorabots, 2008-2021), Chatterbot (Gunther, 2021) y Microsoft Bot Framework (Microsoft, 2019). Por otro lado, las plataformas comerciales no permiten el acceso total al código. En estas plataformas se pierde libertad en la creación del agente conversacional a cambio de datos más eficientes para el entrenamiento y la posibilidad de integrar el *chatbot* en otros productos de la plataforma de forma más sencilla. Ejemplos de plataformas comerciales son Chatfuel (Chatfuel, 2021),

Flow XO (Flow, 2019), Botsify (Botsify, 2021), y Manychat (ManyChat, 2021). En lo referente a la comprensión de lenguaje natural, existen también soluciones comerciales de grandes empresas como Google DialogFlow (Google, 2021), Microsoft LUIS (Microsoft, 2021), Facebook wit.ai (Wit.ai, 2020), Amazon Lex (Amazon, 2021), IBM Watson Conversation (IBM, IBM Watson, 2021) y SAP Conversation AI (SAP, 2021).

2.3.5 Bases de datos de DM para entrenar agentes conversacionales

Tras una búsqueda de bases de conocimiento que sean útiles para entrenar las técnicas de PLN que empleará el agente, no se han encontrado datos que sean potencialmente útiles. Existe un trabajo que empleaba un conjunto de bases de datos basadas en diálogos entre paciente y doctor para crear unos modelos de generación de respuesta automática, pero se ha descartado al no contener información suficiente relativa a la DM (Zeng, y otros, 2020). Otro trabajo (Karami, Dahl, Turner-McGrievy, Kharrazi, & Shaw, 2017) realizaba una búsqueda en Twitter de mensajes escritos (Twits) por usuarios que contuvieran las siguientes palabras: *Diabetes*, *Diet*, *Exercise* y *Obesity*. El estudio observaba la frecuencia de aparición de estas palabras, su correlación y los subtemas más frecuentes que aparecían con estas. Mediante una cuenta de desarrollador de Twitter se ha obtenido acceso a estos mensajes a través de la API de la plataforma. Tras un análisis de estos Twits, se ha observado que la información publicada en la mayoría de los mensajes relacionados con la DM no es útil para el entrenamiento de un agente conversacional, ya que no se asemeja a conversaciones con pacientes diabéticos.

Dado que no se ha encontrado ninguna base de conocimiento que pueda utilizarse para el entrenamiento del agente conversacional, se ha decidido crear una base de datos desde cero. Esta base de datos contendrá un conjunto de diálogos similares a los que habría entre un agente conversacional entrenado para hablar sobre diabetes y un usuario diabético.

2.4 Estado del arte

2.4.1 Creación de bases de datos de DM

Si nos centramos en la creación de bases de datos que contengan información de DM, debemos observar cuál es el foco principal de las predicciones que generan los modelos que se han estudiado hasta ahora. En el artículo de revisión de (Chaki, Ganesh, Cidham, & Theertan, 2020) se seleccionan 107 artículos de modelos predictivos relacionados con

diabetes y se analizan los modelos y las bases de datos utilizadas. En éste, se puede observar que la mayoría de los modelos predictivos tienen dos objetivos distintos. Uno de ellos es el diagnóstico preventivo de la DM (suele ser la DM2) a partir de un conjunto de información de una persona. Para estos modelos se usan bases de datos que contienen información anatómica de la persona y posibles factores que pueden influir en la DM. El otro es la detección de complicaciones derivadas de la DM, concretamente la retinopatía diabética y otras complicaciones oculares. Estos modelos se entrenan con bases de datos formadas por imágenes de ojos sin ninguna patología y ojos con dichas complicaciones. A pesar de que la mayoría de los modelos se centran en esto, también existen modelos que intentan predecir hiperglicemias.

2.4.2 Asistentes conversacionales para la gestión de la DM

Se han encontrado varios estudios centrados en el desarrollo de asistentes virtuales para la gestión de la DM. Esto es debido a que dentro del concepto asistentes virtuales se encuentran principalmente dos tipos distintos de asistentes. El primero consiste en aplicaciones que unifican toda la información relevante relacionada con la DM para facilitar la gestión de la enfermedad. Esta información puede provenir de distintos dispositivos de medida de glucosa o ser introducida manualmente por el usuario o el médico. El segundo se basa en agentes que mantienen conversaciones acerca de la gestión de la DM con un usuario. Estos agentes pueden ser completamente autónomos o ser utilizados por especialistas. Este segundo tipo se alinea más con el objetivo de este proyecto. Seguidamente se muestran varios proyectos relacionados con el segundo tipo de asistentes virtuales para la gestión de la DM.

En el estudio (Looije, Neerincx, & Cnossenc, 2010) se diseñaron tres interfaces tecnológicas distintas para comprobar cuál de ellas era más accesible y confiable para los usuarios. Estas interfaces se pueden observar en la Figura 3. La primera era una interfaz de texto o un chat, la segunda era un avatar virtual con el que podían mantener una conversación y la tercera era un avatar físico con la misma forma que el virtual. Estas tres interfaces hacían unas preguntas a los usuarios y en función de las respuestas se determinaban unos grados de aceptación, confianza, sociales y empáticos. Sin embargo, el asistente no era autónomo, sino que estaba controlado por los experimentadores. Así, este asistente se focalizaba en el grado de aceptación de cada tipo de asistente por parte del usuario, sin tener en cuenta el almacenamiento de datos.

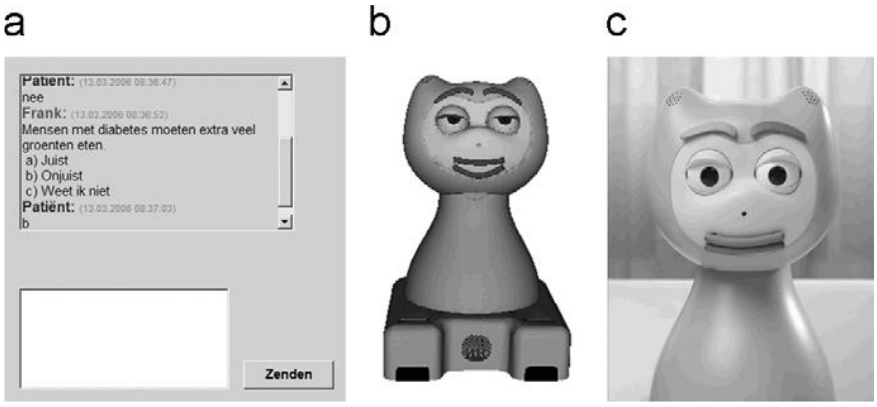

Figura 3: Interfaces distintas del asistente virtual de diabetes. a) Chat, b) Avatar virtual, c) Avatar físico (Looije, Neerincx, & Cnossenc, 2010)

Este mismo equipo de investigación llevó a cabo otro estudio similar utilizando únicamente un robot físico y un avatar digital (Sinoo, y otros, 2018). En este caso, en lugar de usar pacientes de varias edades se probó el asistente únicamente con niños. El objetivo principal era análogo al del previo estudio, observar cuál de los asistentes era más cercano y aceptado por los usuarios. De esta forma, se pretende que los usuarios sigan las recomendaciones de los asistentes con los que más cómodos se sientan. En este caso, se observó que la combinación entre el robot físico y el avatar era la mejor solución.

Otro grupo de investigación creó un asistente robótico para el registro y guía en la gestión de la DM en niños (Al-Taee, Al-Nuaimy, Muhsin, & Al-Ataby, 2017). Este robot estaba conectado con un sistema que a su vez se conectaba con los especialistas médicos (Figura 4). Este sistema podía mostrar información acerca del plan médico y características físicas del paciente, y a los cuidadores les mostraba un indicador que determinaba la gestión de la glucosa del paciente. Así, en caso de tener niveles altos o bajos el responsable podía actuar de forma rápida. El asistente también almacenaba la información diaria del control de la diabetes y la mostraba de formas distintas. La información acerca de los niveles de glucosa se transmitía de los sensores al asistente directamente, y éste gestionaba esta información para sugerir una gestión óptima de la medicación. Para fomentar la confianza con el usuario, también empleaba un diálogo específico y adaptado al paciente. También tenía la posibilidad de almacenar información y mensajes de voz del paciente. Toda la información almacenada por el asistente se subía a un servidor en internet, al cual podían acceder tanto el médico, como el niño diabético, como la persona responsable del niño. El robot también obtenía la información acerca de los diálogos que tenía que realizar de este mismo servidor en internet. Sin embargo, el sistema de diálogos estaba definido por nodos que correspondían a interacciones, y cada interacción correspondía con una acción del robot. Este asistente estaba más centrado en la coordinación entre paciente, responsable y médico y la mejora en la gestión de la enfermedad por parte de todos.

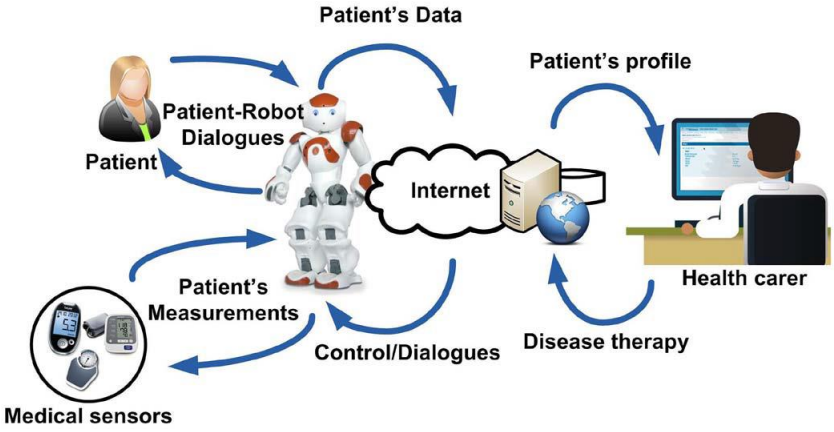

Figura 4: Funcionamiento del sistema relacionado con el asistente robótico para niños (Al-Taee, Al-Nuaimy, Muhsin, & Al-Ataby, 2017).

En 2019 se diseñó un asistente virtual antropomórfico dentro del proyecto VASelfCare que daba soporte a enfermos de DM tipo 2 mayores de 65 años (Magyar, y otros, 2019). Este se diseñó para dispositivos portátiles como puede ser un móvil o una Tablet. El objetivo principal de este asistente era producir un cambio en el estilo de vida del enfermo que contribuya a una mejor gestión de la DM2. Para ello, el asistente se focalizaba en promover la auto medicación, la actividad física y un control de la dieta. Este agente conversacional antropomórfico se denominó Vitória y su interfaz gráfica se puede observar en la Figura 5. Este *bot* se comunicaba con el usuario por voz y texto, y al hacerlo mostraba una respuesta emocional en función del input del usuario. De esta forma, se definía al asistente como un agente relacional que pretendía establecer una unión con el usuario para que siguiese los consejos del agente en lo referente al control de la DM2.

Figura 5: Interfaz gráfica del asistente virtual (Vitória) VASelfCare (Magyar, y otros, 2019)

La conversación no era la única entrada de información para este asistente virtual, que también almacenaba el número de pasos o la medicación recibida. Esta información se introducía manualmente por el usuario, que obtenía recomendaciones en función de los valores introducidos. La aplicación también tenía la posibilidad de planificar la semana con actividades y dietas, observar datos de interés relacionados con el usuario e información detallada acerca de la DM2 y los factores que influyen en su control. Toda esta información la recibía un módulo conversacional, y posteriormente se almacenaba en formato .json en una base de datos distinta para cada usuario. La propia aplicación utilizaba también esta información para decidir el diálogo que llevaba a cabo el asistente, el cual se iba mejorando mediante el uso de técnicas de aprendizaje por refuerzo. Por tanto, a pesar de que este agente almacenaba información, esta se usaba únicamente para adecuar la conversación con el usuario y hacer mejores recomendaciones.

Sin embargo, no todos los asistentes virtuales relacionados con la DM tienen como objetivo la gestión de la enfermedad. También se diseñaron asistentes virtuales para la detección y diagnóstico de la enfermedad (Spänig, et al., 2019). Este asistente consistía en una cabina con distintos aparatos de medida que registraban y calculaban datos del paciente como el peso, altura e índice de masa corporal entre otros. Además de esta información, el asistente también contenía un módulo conversacional mediante el cual realizaba unas preguntas y almacenaba y procesaba las respuestas de voz recibidas por el usuario. Estas preguntas hacían referencia a posibles síntomas relacionados con la DM. Toda esta información recibida se introducía en modelos de redes neuronales profundas y en modelos de máquinas de vector soporte, que devolvían un diagnóstico haciendo análisis probabilísticos de los datos. El asistente establecía una probabilidad de padecer la enfermedad y a medida que iba adquiriendo más información iba modificando la probabilidad. Finalmente, emitía un informe con la información adquirida y el diagnóstico realizado.

Por tanto, podemos concluir que la mayoría de los asistentes conversacionales estudiados tienen como objetivo una mejor gestión de la DM. El objetivo final de estos es establecer una relación con el usuario y generar cambios de comportamiento en enfermos de DM. Con esto también buscan mejorar la conexión entre el enfermo y el médico, generando predicciones que puedan ser de utilidad para ambos. Esto difiere del objetivo del proyecto actual, que es la creación de bases de datos con información sobre la DM que puedan ser estandarizadas y usadas de forma global.

2.5 Conclusiones parciales

Para el desarrollo de esta herramienta se empleará la API de Telegram, que es una aplicación que se utiliza como infraestructura para la creación y despliegue de *chatbots*. En la actualidad, Telegram es una aplicación de mensajería muy utilizada, por debajo de otras muy conocidas como WhatsApp o Facebook Messenger. La integración de un agente conversacional en Telegram se realiza de forma muy sencilla gracias a que esta aplicación de mensajería está muy alineada con el despliegue de *chatbots* de todo tipo. De esta forma, el único requerimiento necesario para poder usar el agente conversacional será tener instalada la aplicación. Esto hace que se pueda acceder a la herramienta de forma mucho más simple y cómoda para los usuarios. Además de Telegram, también se podría desarrollar en otras aplicaciones de mensajería, pero inicialmente se decide utilizar únicamente Telegram para reducir el número de peticiones simultáneas y el almacenamiento de registros de conversaciones.

Por otro lado, la infraestructura que se utilizará para desarrollar el agente conversacional es DialogFlow, un *framework* diseñado por Google. DialogFlow comparado con otras soluciones de fuente abierta ofrece muchas posibilidades para la creación del agente de una forma muy sencilla. Además, esta plataforma contiene un conjunto de paquetes que pueden ser instalados en el *chatbot* para gestionar determinados temas o partes de una conversación. Esto permite que el diseño de la herramienta se centre en la conversación relacionada con la DM y que el resto de los elementos de la conversación se puedan gestionar de forma más sencilla. Esta plataforma también tiene una fácil integración con una base de datos (Firebase) que facilita el almacenamiento de la información obtenida en la conversación y la recuperación de esta en futuras conversaciones. Esto es una gran ventaja frente a las plataformas de fuente abierta, en las cuales hay que buscar una base de datos externa y conectarla con el agente conversacional.

Para este proyecto se pretende utilizar los métodos de creación de *chatbot* relacionados con *embeddings* y modelos de aprendizaje automático, ya que dotan al agente de una mayor capacidad de comprensión de texto en comparación con los métodos basados en reglas. Estos últimos suelen ser mejores cuando una conversación sigue siempre una misma estructura, pero se pretende crear un agente conversacional que sea cercano al enfermo de DM y que tenga en cuenta el histórico de conversaciones, adaptándose a cada caso. Para ello, el agente creado debe ser capaz de obtener información relevante de la DM y adaptarse a nuevos diálogos basado en un histórico de conversaciones con pacientes. Finalmente, se pretende implementar una metodología para crear respuestas de texto creadas por un modelo de GLN. Para el entrenamiento de este modelo se creará una base de datos inicialmente

simple con información básica relacionada con la DM, que se irá ampliando con los registros de conversaciones del agente.

3 Objetivos y metodología de trabajo

3.1 Objetivo general

El objetivo de este proyecto es la creación de un agente conversacional centrado en el almacenamiento de datos relacionados con la DM1. Este agente utilizará técnicas de PLN para ser capaz de entender los mensajes que el usuario envíe en la conversación, responder de forma conveniente y almacenar la información relevante sobre la DM1 que esos mensajes contengan. Finalmente se evaluarán las capacidades funcionales de la herramienta desarrollada y la frecuencia de uso por parte de los usuarios.

3.2 Objetivos específicos

Los objetivos específicos que se pretenden cumplir en este proyecto son principalmente cuatro.

- Estudiar el estado del arte de agentes conversacionales centrados en DM. Dentro de este se incluyen:
    - Explorar las distintas infraestructuras para la creación de *chatbots*.
    - Identificar los datos más relevantes y comunes en la gestión de la DM.
    - Determinar la diferencia entre información útil para la DM y la información irrelevante.
- Diseñar un agente conversacional de forma óptima haciendo uso de toda la información estudiada.
- Implementar el agente utilizando las técnicas, plataformas y herramientas descritas anteriormente.
- Evaluar el funcionamiento de la herramienta desarrollada y su frecuencia de uso.

3.3 Metodología del trabajo

Para la creación de esta herramienta se seguirá una metodología iterativa e incremental. Este tipo de metodología de desarrollo de software se basa en la creación de un modelo con un conjunto de especificaciones iniciales que se va evaluando y expandiendo hasta crear un producto final. No se define un número de evaluaciones previamente, sino que se va

desarrollando de forma iterativa y se evalúa tras cada ciclo para modificar las especificaciones hasta que el modelo se convierte en una aplicación totalmente funcional. La Figura 6 muestra la estructura de este tipo de metodología, en la cual se observan algunos pasos que realizan una única vez y otros que se realizan de forma iterativa (Despa, 2014).

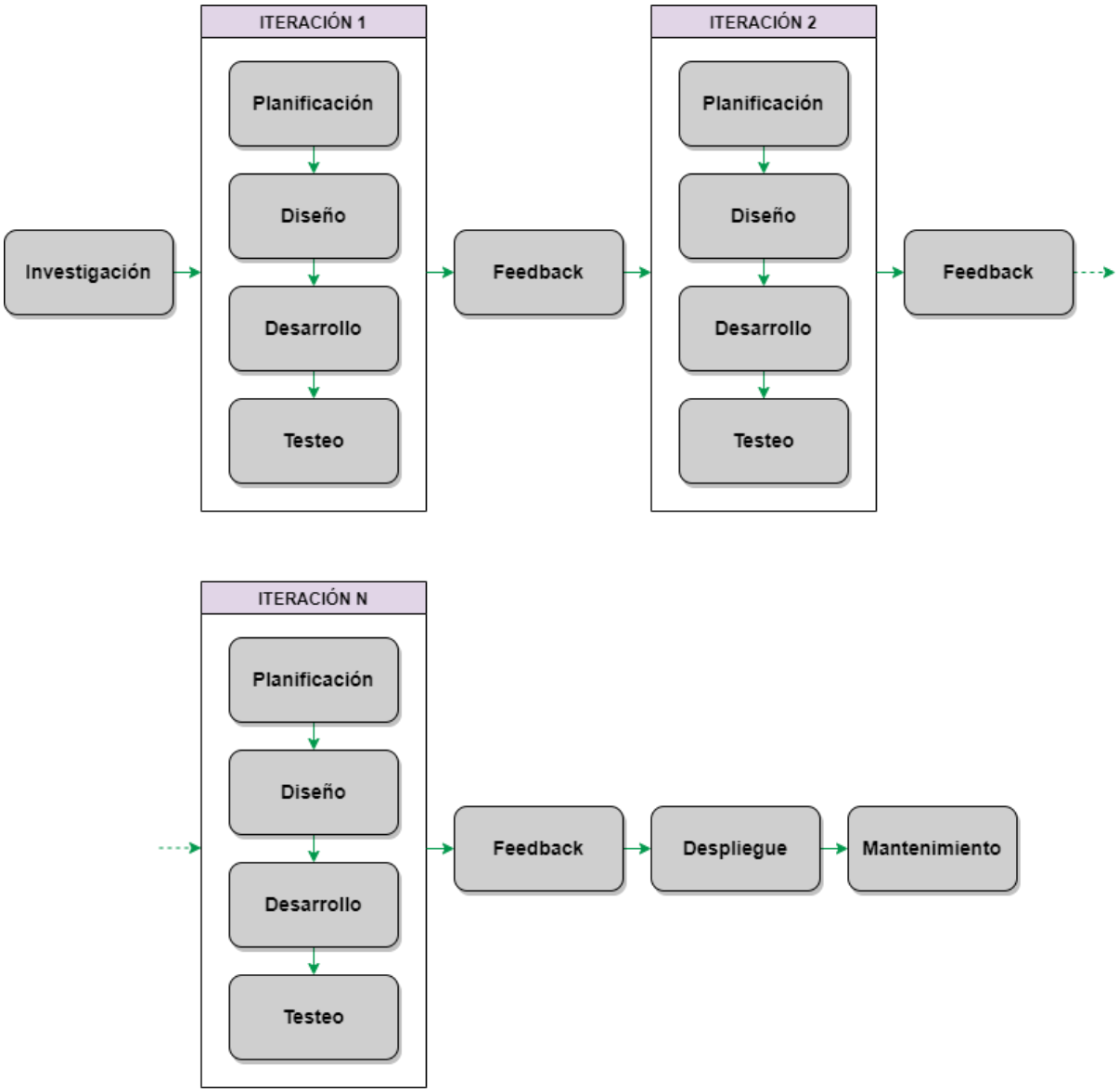

Figura 6: Metodología de desarrollo de software iterativa e incremental (Despa, 2014)

El primer paso que se realiza en este tipo de proyectos se denomina investigación, y en este proyecto se ha realizado en los apartados anteriores. Inicialmente se han establecido los requerimientos básicos y los objetivos que nuestra herramienta de software debe cumplir. Posteriormente se ha hecho un estudio de toda la documentación necesaria para desarrollar un agente conversacional de este tipo. Finalmente, se han realizado unas conclusiones parciales en las cuales se han decidido los distintos recursos, infraestructuras y técnicas con las cuales se desarrollará la aplicación para cumplir con los objetivos establecidos. Se ha

decidido utilizar Dialogflow como herramienta para el desarrollo del agente conversacional, Telegram como interfaz para mantener una conversación con el usuario, y Firebase como base de datos donde almacenar la información sobre DM recopilada.

Una vez realizado todo el estudio del contexto y estado del arte, se sigue un conjunto de etapas para crear un modelo que cumpla con las especificaciones iniciales que se han definido tras la investigación. El modelo creado se evaluará posteriormente para comprobar si cumple con los requerimientos y realizar las modificaciones necesarias, además de observar qué nuevas funcionalidades o características se pueden añadir para crear un producto final completo. Todo este proceso se repetirá el número de veces que sea necesario hasta obtener un producto final. El conjunto de etapas que se realizan de forma iterativa son planificación, diseño, desarrollo y testeo, que irán seguidas de una etapa en la cual se analizará el *feedback* recibido.

El primer paso en la planificación es la definición de un flujo general para la aplicación, que posteriormente se irá dividiendo en procesos menores con un número de funcionalidades concretas. En función de estas se diseñará la estructura de la base de datos y se decidirá la tecnología a utilizar. Para la creación de un agente conversacional que recopile información relacionada con la DM, el primer paso es que el agente sea capaz de mantener una conversación cuotidiana con un usuario. En estas conversaciones se debe recopilar información que sea almacenada en una base de datos, que sirva para que las conversaciones futuras sean más naturales. Dentro de estas conversaciones habrá información relacionada con la DM, ya sea porque el agente haga preguntas o bien que el usuario quiera dar esta información. Esta información también se almacenará en la base de datos de forma distinta a la información básica de cada usuario. Toda esta conversación se llevará a cabo a través de una interfaz gráfica que será Telegram, en forma de texto escrito. Por tanto, es necesario conectar todos los elementos que formen el agente conversacional: la interfaz gráfica, el desarrollo del agente conversacional y la base de datos. En la base de datos (Firebase) se debe definir la estructura de los datos que se almacenarán, que serán en formato .json. Esta estructura será distinta para la información de los usuarios y para la información relacionada con la DM. Para que el agente pueda mantener conversaciones se utilizarán librerías de Small Talk, que se adaptarán y se añadirá más profundidad a la funcionalidad básica. Además de esta, se añadirán hilos de conversación específicos para los elementos que más influyen en la DM1: el nivel de glucosa en sangre, la ingesta de alimentos, el ejercicio físico, la cantidad de insulina y situaciones que causen estrés al usuario.

La etapa de diseño suele estar centrada en la apariencia gráfica, que es la parte de la aplicación que el usuario observa y con la cual interactúa. En este caso la interfaz gráfica de

la aplicación será una conversación de texto utilizando la aplicación de mensajería Telegram. Una mayor capacidad de respuesta por parte del agente o el uso de información almacenada para una conversación más natural debería mejorar la experiencia del usuario, pero esto forma parte del desarrollo del agente.

En el desarrollo de la aplicación es cuando se crea el código y el software propiamente dicho. Se utilizará el *framework* de Dialogflow para desarrollar el agente e implementar nuevas posibilidades de conversación, que se irán guardando de forma automática. El elemento básico que utiliza un agente para mantener una conversación se denomina *intent*. Un *intent* se puede considerar una intención detrás de una oración escrita por el usuario. Utilizando técnicas de PLN, el agente es capaz de asignar un *intent* a una línea de conversación escrita por el usuario y tiene configuradas unas respuestas concretas para cada *intent*. Dentro de un *intent* el agente puede identificar palabras o expresiones de importancia especial, las cuales almacenará en una variable. Estos se denominan parámetros, y se pueden almacenar en la base de datos para acumular información importante de la conversación. Esta información podrá ser posteriormente recuperada para generar respuestas más parecidas a un ser humano o para comprobar que el parámetro se ha guardado correctamente. Pese a que toda esta información se vaya guardando a tiempo real, es posible guardar versiones del agente conversacional. En esta fase también se implementará el proceso de almacenamiento de información en la base de datos y la creación de las respuestas dinámicas. Para ello se entrenará un modelo de GLN capaz de generar respuestas a mensajes de usuario con información sobre DM.

Después de la etapa de desarrollo es necesaria una evaluación de la aplicación desarrollada, y esto se realiza en la etapa de testeo. Para testear un agente conversacional se comprueba que es capaz de identificar todos los *intents* que se han definido de forma correcta y que responde de forma adecuada. También se debe evaluar la usabilidad del agente de forma estandarizada. Esta etapa no debe ser realizada únicamente por desarrolladores, sino que también la puede realizar un pequeño grupo de usuarios. Estos usuarios ofrecerán situaciones más reales, mantendrán conversaciones distintas y añadirán más variabilidad a la hora de testear el agente.

La etapa de testeo dará como resultado una aplicación mejorada y un *feedback*, que contiene un conjunto de elementos o funcionalidades que la aplicación no posee y debería de tener. Aquí se observarán los registros de conversaciones del agente con los usuarios y se comprobará que hilos de conversación no ha sido capaz de seguir y por tanto serían necesarios para el agente. También se comprobará el funcionamiento del modelo de GLN, el cual se reentrenará en caso necesario. Una vez se haya evaluado el resultado, se volverá a

empezar una etapa de planificación para añadir las nuevas funcionalidades y se seguirá con la etapa de desarrollo y testeo respectivamente. Estos pasos se repetirán de forma cíclica hasta que la aplicación sea capaz de mantener conversaciones totalmente completas y no se detecte ninguna carencia.

Una vez se considere que la herramienta de software está finalizada, se procede a su instalación en un ambiente real. Esta etapa es trivial en el caso de un agente conversacional creado utilizando Dialogflow, debido a que ya estará integrado en la aplicación de mensajería y será accesible por cualquier usuario en todo momento.

Finalmente existe una etapa de mantenimiento, en la cual se va controlando la aplicación para que funcione siguiendo el comportamiento esperado en todo momento. En un agente conversacional, el objetivo principal de esta etapa es comprobar que el agente entiende correctamente los mensajes y es capaz de responder de forma adecuada. En los agentes que aprenden en base a las conversaciones realizadas, hay que impedir que numerosas conversaciones inadecuadas influyan negativamente en el comportamiento del agente. Por tanto, hay que asegurarse de que el agente cumple con la función establecida inicialmente y de que su comportamiento no ha divergido debido a su uso.

4 Identificación de requisitos

La identificación de requisitos de esta herramienta se ha dividido en dos apartados principalmente: unos requisitos globales y otros más específicos. Los requisitos globales hacen referencia a funcionalidades que el agente conversacional debería tener al finalizar el proyecto. Los requisitos específicos son el conjunto de pasos que se han de llevar a cabo en cada una de las plataformas elegidas para poder empezar con el desarrollo de la aplicación.

4.1 Requisitos globales del proyecto

De forma global, esta herramienta de software debe cumplir con un conjunto de requerimientos. El agente debería ser capaz de mantener una conversación cercana con el usuario, en la cual se hable de situaciones diarias. Esto es especialmente importante para que la herramienta software se utilice regularmente y la información de la DM se recopile de forma natural y no forzada. Para ello, el agente debe recopilar información acerca del usuario y utilizarla en futuras conversaciones, haciendo que la interacción sea más natural. El agente también ha de ser capaz de almacenar cualquier información relacionada con la DM en una conversación. Esta información ha de ser almacenada de una forma ordenada y correcta, para que pueda ser utilizada en el futuro en caso de que se avance este proyecto en otro más complejo. Por último, el agente utilizará técnicas de PLN para la comprensión de los mensajes del usuario y responderá respuestas predefinidas. Posteriormente se deben configurar las respuestas dinámicas creadas mediante técnicas de GLN.

Finalmente, no toda la información que la herramienta recopile será compartida. Toda la información personal de los usuarios que el agente recoja será únicamente usada para mantener conversaciones más cercanas con el usuario. La información relacionada con la DM que se almacene no contendrá ningún dato personal acerca del usuario, y únicamente estará relacionada con la gestión de la enfermedad.

4.2 Requisitos específicos de las herramientas utilizadas

Cómo se ha comentado anteriormente, para el desarrollo de esta herramienta software se utilizarán tres aplicaciones: Telegram, Dialogflow y Firebase. Telegram es la aplicación de mensajería donde los usuarios podrán mantener conversaciones de texto con el agente. Dialogflow es la plataforma de desarrollo de *chatbots* que se utilizará y, por tanto, es la que contiene todo el funcionamiento interno del agente conversacional. Firebase es la base de datos donde se almacenará toda la información recibida en las conversaciones y se extraerá

para que el agente realice respuestas más complejas. Para poder conectar estas tres herramientas y finalmente empezar a desarrollar el agente, es necesario llevar a cabo unos procesos en cada una de ellas.

La creación de un nuevo *bot* en Telegram se realiza mediante Fatherbot, que es un *bot* que sirve como guía para gestionar cualquier nuevo agente que quiera integrarse con la aplicación. Utilizando Fatherbot se define el nombre e ID del nuevo agente y recibimos un token único que se usará en el *framework* de desarrollo del *chatbot* para conectar con Telegram. Posteriormente se pueden también cambiar algunas configuraciones de la interfaz gráfica del agente mediante un conjunto de comandos. Este es el único requerimiento para tener un agente conversacional en esta aplicación de mensajería.

Para poder crear un agente en Dialogflow y una base de datos en Firebase es necesario primero crear un proyecto nuevo en Google Cloud Platform. Esta plataforma unifica todas las aplicaciones de desarrollo web que ofrece Google y permite la gestión de proyectos. Una vez creado un proyecto en esta plataforma, es necesario elegir un plan de facturación. Inicialmente se elegirá un plan gratuito, que será suficiente para el despliegue inicial del agente conversacional.

Después de seleccionar el proyecto de Google Cloud Platform en Firebase, podremos crear una base de datos Firestore. Esta estará estructurada en colecciones, y para cada elemento de una colección se recopila el mismo tipo de datos. Aquí se deberán crear distintas colecciones para los distintos tipos de información que se pretende almacenar de cada usuario.

5 Descripción de la herramienta software desarrollada

5.1 Creación del agente en Dialogflow

El primer paso para la creación de un agente conversacional es hacer *login* dentro de la web de Dialogflow utilizando una cuenta de Google. Una vez dentro, hay que hacer clic a crear agente y aparece la pantalla de la Figura 7. En ella se debe escribir el nombre del agente, el idioma que utilizará por defecto y la zona horaria en la cual se encuentra. Para este proyecto se ha elegido como idioma por defecto el inglés debido a que existe mucha más documentación y recursos para entrenar modelos de generación de texto en este idioma. También se ha añadido el español como idioma, pero al no ser el idioma por defecto del agente, únicamente se podrá hablar en español utilizando el chat integrado de Dialogflow.

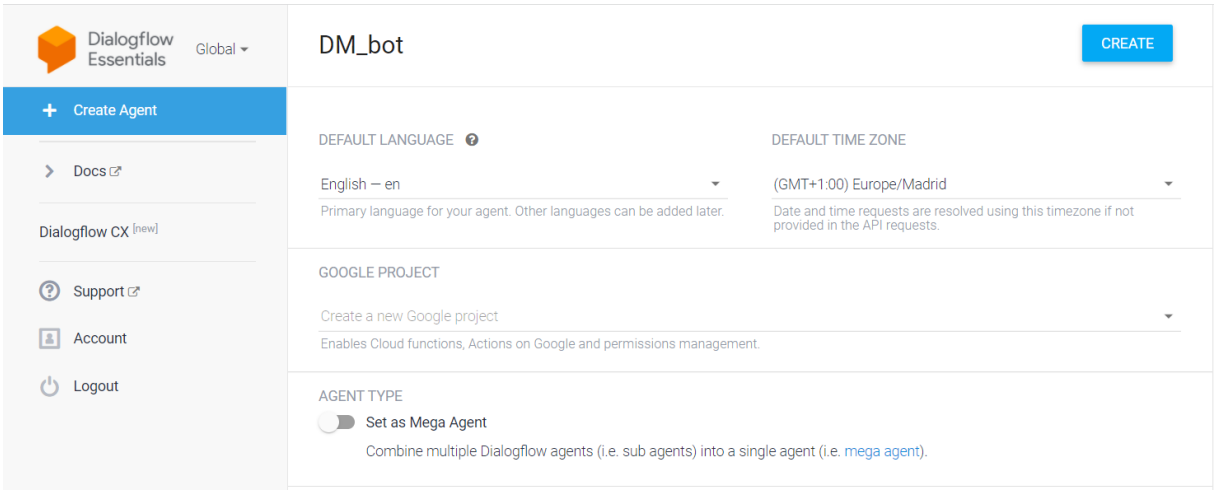

Figura 7: Creación del agente conversacional en Dialogflow.

Al crear el nuevo agente se crea automáticamente un nuevo proyecto de Google Cloud Functions, que servirá para conectar con la base de datos. En el agente aparecen dos *intents* por defecto: el saludo y el *fallback*, que es el mensaje que envía el agente cuando no es capaz de asociar un mensaje del usuario con ninguno de los *intents* definidos. En el lado derecho aparece una consola mediante la cual se pueden mantener conversaciones con el agente, cuya utilidad es principalmente para llevar a cabo pruebas y testeos. En el menú del lado izquierdo aparece el nombre del nuevo agente y un conjunto de opciones con las cuales se puede desarrollar la herramienta. Las más relevantes y que por tanto más se han usado durante el desarrollo del agente conversacional son los *intents*, las entidades, el *fulfilment* y las integraciones.

5.1.1 Intents

Los *intents* de un agente son las diferentes intenciones en las cuales se puede clasificar un mensaje del usuario. Para poder mantener una conversación con el agente se debe definir una multitud de *intents* y, para cada mensaje que el usuario envía, Dialogflow lo hace coincidir con el *intent* más parecido y devuelve una respuesta. Un *intent* contiene cuatro elementos básicos: frases de entrenamiento, acciones, parámetros y respuestas.

Las frases de entrenamiento son un conjunto de ejemplos de lo que el usuario podría escribir al querer transmitir un mensaje. La coincidencia de un mensaje del usuario con un *intent* se realiza utilizando métodos de *embeddings*, trabajando de forma similar a un modelo de clasificación donde las clases son el conjunto de *intents*. Cuando un mensaje coincide con un *intent*, el agente puede llevar a cabo una acción como podría ser, por ejemplo, una llamada a una API para realizar una reserva. Los parámetros son información definida previamente que se extrae del mensaje del usuario y que se puede utilizar en la conversación en un futuro. Por último, se deberían escribir una o más respuestas que el agente dará en la conversación cuando se detecte el *intent*.

5.1.2 Entidades

Tal y como se expresa en la documentación de Dialogflow: “cada parámetro de intent tiene un tipo, denominado *tipo de entidad*, que determina de forma exacta cómo se extraen los datos de una expresión de usuario final” (Google, Entidades | Dialogflow ES | Google Cloud, 2021). Por tanto, se podría decir que una entidad es una estandarización en la forma de obtener una información concreta de un mensaje del usuario. Existen unas entidades definidas por Dialogflow como son números, colores, tiempo, fechas, nombres o direcciones de correo electrónico entre otras. En caso de que ninguna de las entidades del sistema sea útil, es posible crear entidades personalizadas para extraer la información deseada de los mensajes. Una entidad personalizada tiene un tipo y un número de entradas. El tipo de entidad se corresponde con el tipo de información que se pretende extraer, y las entradas son los distintos valores posibles que puede tener una entidad. Si se admiten sinónimos para las entradas, hay que definir un valor de referencia para cada una. También se puede habilitar la expansión automática de una entidad con valores no definidos inicialmente, o activar una opción para que expresiones o palabras mal escritas sean reconocidas como un valor de la entidad. También existen las expresiones regulares, que son entidades formadas por números, letras o ambos y siempre tienen la misma estructura, como podría ser un número de teléfono o una matrícula de un vehículo.

5.1.3 *Fulfillment* e integraciones

El *fulfillment*, también denominado entrega, se puede activar en cada *intent* para hacer que el agente proporcione una respuesta dinámica. En lugar de dar una respuesta estática que se ha definido en el apartado *intents*, con esta opción el agente puede realizar una llamada a un servicio de *webhook* externo para llevar a cabo cualquier acción. En este proyecto el servicio de *webhook* externo se encuentra desplegado en Firebase Functions. Las acciones que se llevan a cabo son el almacenamiento y recuperación de información relacionada con la DM y la generación de respuestas dinámicas utilizando un modelo que se ejecuta en la máquina local. En la Figura 8 se puede observar el flujo de funcionamiento de las entregas en Dialogflow (Google, Fulfillment | Dialogflow ES | Google Cloud, 2021).

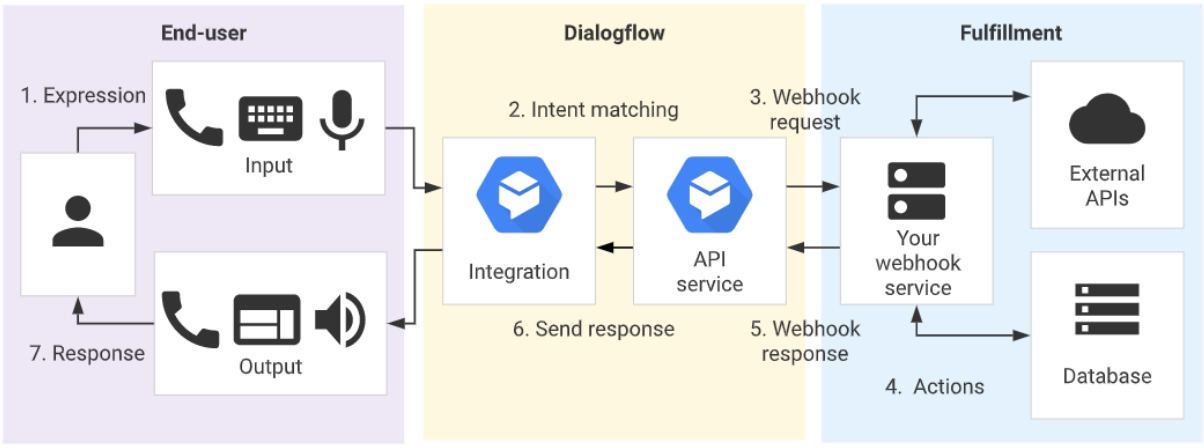

Figura 8: Flujo de funcionamiento de un intent con la entrega habilitada (Google, Fulfillment | Dialogflow ES | Google Cloud, 2021).

La integración consiste en el despliegue del agente conversacional creado mediante Dialogflow a una plataforma de conversación. Estas plataformas pueden ser de conversación telefónica o escrita, y algunas de ellas tienen compatibilidad total con Dialogflow mientras que otras requieren de una integración de código abierto.

5.2 Creación del *bot* en Telegram

La creación de un nuevo agente conversacional en Telegram se realiza mediante BotFather, que es un *bot* creado por la aplicación de mensajería que gestiona la creación y modificación de cualquier *bot* dentro de Telegram. Para iniciar una conversación con BotFather hay que enviar el comando /start, con el cual recibimos un listado de comandos y funciones que se pueden realizar. Utilizando el comando /newbot da comienzo una conversación en la cual se establece el nombre y nombre de usuario del nuevo agente conversacional. Finalmente, BotFather envía un Token que es un identificador único del *bot* recién creado y servirá a

cualquiera que lo tenga para controlarlo. El mensaje enviado por BotFather que contiene el Token se puede observar en la Figura 9.

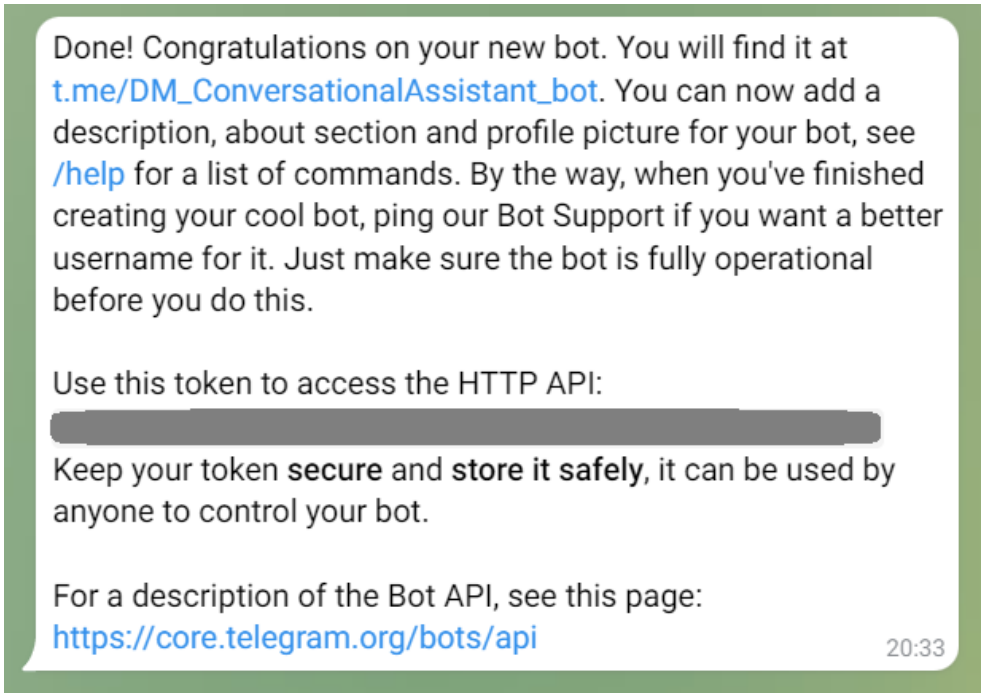

Figura 9: Mensaje final de BotFather en la creación de un nuevo bot en Telegram.

Después de obtener el Token hay que volver a Dialogflow y abrir el apartado de integraciones. Dentro de esta sección están las diferentes aplicaciones de telefonía, mensajería basada en texto y aplicaciones de fuente abierta con las cuales se puede conectar el agente conversacional. Tras hacer clic en Telegram, Dialogflow pide que se introduzca el Token que corresponde con el agente creado y que se pulse START para realizar la integración. Para comprobar que la integración se ha realizado correctamente únicamente es necesario iniciar una conversación con el nuevo agente mediante Telegram y recibir una respuesta.

5.3 Configuración de la base de datos y de las entregas

Para poder establecer una base de datos en la cual almacenar la información relacionada con la DM, es necesario hacer *login* en la web de Firebase con la misma cuenta de Google con la que se ha creado el agente de Dialogflow. Al crear un nuevo proyecto en Firebase, se debe escoger el mismo proyecto de Google Cloud Functions que se creó en la sección 5.1. Esto permitirá la conexión entre ambos programas y por tanto hará que sea posible la transmisión de información. Una vez abierto el proyecto de Firebase, existen dos tipos de bases de datos para crear: Firestore Database y Realtime Database. A priori ambas bases de datos parecen adecuadas para el nuevo agente conversacional, pero se decide utilizar Firestore Database debido a que es una versión nueva de Realtime Database, tiene un modelo de datos más

intuitivo y permite realizar consultas más rápidas. Al crear la base de datos en Firestore Database se selecciona modo de producción y se escoge el servidor donde se debe alojar. Firestore Database es una base de datos NoSQL que se organiza en colecciones que contienen documentos, y cada documento contiene un conjunto de pares clave-valor.

Una vez creada la base de datos, es necesario activar las entregas en Dialogflow para permitir la comunicación entre ambas plataformas. Esto se realiza mediante la activación del editor en línea, que se encuentra en el menú de *fullfillment*. Al activar el editor en línea aparece una ventana indicando que es necesario activar un plan de facturación en Google Cloud Console. Google Cloud Console es la plataforma desde la cual se puede gestionar todo lo relacionado con el proyecto de Google Cloud Functions. Para este proyecto se ha escogido inicialmente un plan de facturación gratuito, y en caso de realizar un número de llamadas a la base de datos mayor de lo permitido por este plan, se escogería otro plan de pago.

Existen dos posibilidades para trabajar con entregas: el editor en línea o un servicio de *webhook* externo. El editor en línea permite la modificación del código de entregas en la misma interfaz web del agente en Dialogflow, pero tiene ciertas limitaciones a la hora de escribir código. Esto se debe a que este editor despliega a Cloud Functions, que utiliza Node 6. Esta versión es relativamente antigua y no permite el uso de algunas funcionalidades actuales de este entorno, haciendo que el trabajo con el editor en línea sea más difícil. Por tanto, se ha decidido habilitar el servicio de *webhook*, que al estar conectado con el mismo proyecto de Google Cloud añade automáticamente la dirección de Firebase Functions donde se ejecuta el código. Para poder trabajar con *webhook* ha sido necesario descargar el código del editor en línea, descargar la API de Firebase en la máquina local e instalarla. Una vez instalada ha sido necesario inicializarla especificando todas las funciones de la API que se iban a utilizar y configurándolas.

Al inicializar la API, se ha creado automáticamente una carpeta *functions* dentro del directorio donde se ha trabajado con el proyecto en local. Este directorio completo se puede observar en el Anexo I. Cada vez que se han realizado cambios en las funciones de las entregas ha sido necesario desplegar la carpeta *functions* a Firebase para aplicarlos, debido a que esta carpeta contiene los mismos archivos que había en el editor en línea: index.js y package.json. El archivo index.js contiene todo el código de entregas, y es con el que se trabaja principalmente en este proyecto. Para poder tener acceso a la base de datos es necesario inicializarla y definirla como una constante. Esto lo hemos hecho añadiendo las constantes admin y db e inicializando Firestore como se muestra en el Código 1.

```
"use strict";

const functions = require("firebase-functions");
const admin = require("firebase-admin");
const {WebhookClient} = require("dialogflow-fulfillment");

const rp = require("request-promise");

process.env.DEBUG = "dialogflow:debug"; // enables lib debugging state-
ments
admin.initializeApp(functions.config().firebase);
const db = admin.firestore();
```

Código 1: Código en lenguaje Node.js utilizado para inicializar y definir la base de datos Firestore.

El mapeo de los *intents* a sus funciones correspondientes se realiza estableciendo el denominado intentMap. En éste se deben escribir el nombre del *intent* de Dialogflow y el nombre de la función que se ejecuta cuando el *intent* es identificado. En el Código 2 se puede observar un ejemplo de cómo se define en index.js el intentMap en un agente con dos *intents* con entregas.

```
let intentMap = new Map();
intentMap.set('Welcome Intent', welcome);
intentMap.set('Fallback Intent', welcome);
agent.handleRequest(intentMap);
```

Código 2: Mapeo de los intents con las funciones.

El archivo package.json contiene la información básica del *fulfillment* e incluye las dependencias necesarias para el correcto funcionamiento de las entregas, que se pueden observar en el Código 3. El cambio principal frente al documento que venía por defecto es instalar la versión 0.6.1 de dialogflow-fulfillment, que permite una correcta comunicación entre el Dialogflow y la base de datos.

```
  "dependencies": {
    "actions-on-google": "^2.5.0",
    "cors": "^2.8.5",
    "dialogflow": "^0.6.0",
    "dialogflow-fulfillment": "^0.6.1",
    "firebase-admin": "^10.0.1",
    "firebase-functions": "^3.16.0",
    "request": "^2.88.2",
    "request-promise": "^4.2.6"
  },
```

Código 3: Dependencias deseadas dentro de package.json.

5.4 Importación de Smalltalk y configuración del *intent* de saludo

Dialogflow contiene un conjunto de agentes prediseñados especializados para diferentes tipos de conversación, que se pueden crear como agentes nuevos o importar al agente actual. Dependiendo del idioma configurado para el agente existen más o menos agentes prediseñados que se pueden importar de forma completa o parcial. En el caso de este proyecto se pretende que el agente mantenga conversaciones principalmente en inglés, pero también en español, y para el español únicamente existen cuatro agentes prediseñados disponibles para importar. Uno de ellos es el agente Smalltalk, que contiene *intents* correspondientes a temas de conversación simples que hacen al agente un poco más personal. Smalltalk es de gran utilidad para el agente que se ha desarrollado en este proyecto y por tanto se ha importado. Al importar Smalltalk a un agente cuyo idioma es el español se añaden automáticamente un conjunto de *intents* sobre temas simples, pero no se añaden las respuestas que realizará el agente a cada *intent*. Por ello, se deben escribir las respuestas una a una en el apartado *intents*, y si se escribe más de una para un mismo *intent*, cuando el agente deba responder escogerá una aleatoriamente entre todas las respuestas posibles. Para la elaboración de las respuestas en este proyecto se han usado como ejemplo las frases de respuesta del agente Smalltalk en idioma inglés, que está totalmente implementado.

```
  function welcome(agent) {
    // Get user ID from Telegram payload data
    const userId = agent.originalRequest.payload.data.from.id;

    // Check if the database exists. If it does it is a known user, if it
doesn't it is a new user.
    return db.collection(""+userId).doc("Basic Info").get()
        .then((doc) => {
          if (!doc.exists) {
            agent.add("Hi, nice to meet you. What's your name?");
          } else {
            const user_name = doc.data().Name;
            agent.add(`Hi again ${user_name}, how are you today?`);
          }
          return Promise.resolve();
        }).catch(() => {
          agent.add("Error reading entry from the Firestore database.");
        });
  }
```

Código 4: Función de saludo del agente.

Como se ha comentado anteriormente, al crear un nuevo agente se crean por defecto los *intents* de saludo y *fallback*. Para el *intent* de *fallback* únicamente se han añadido respuestas personalizadas. Sin embargo, en el *intent* de saludo se ha habilitado la entrega y se ha configurado para realizar un saludo distinto si es la primera conversación con un usuario o bien si ya ha habido una conversación con el usuario previamente. El funcionamiento básico del agente cuando se detecta el *intent* de saludo se puede observar en el Código 4, y se basa en comprobar si el nombre del usuario existe en la base de datos. En caso de que no exista, se inicia una conversación de introducción para recopilar y almacenar unos datos iniciales sobre el usuario. Si por el contrario el usuario existe en la base de datos, se realiza un saludo personalizado para ese usuario.

5.5 Conversación de introducción

Cuando un usuario interactúa por primera vez con el agente se inicia una conversación de introducción. Esta conversación tiene una duración de 6 turnos y se basa en un seguido de preguntas que el agente realiza al usuario para almacenar una información básica. La información básica recopilada es la siguiente: nombre del usuario, su edad actual, la edad en la cual le diagnosticaron la DM, el tipo de DM y el tratamiento que sigue para la misma. Se puede observar un ejemplo de esta conversación en la Figura 10. Para completar esta información se requiere crear los *intents* correspondientes, crear entidades en caso necesario y configurar las funciones asociadas en el archivo index.js.

Se han creado 5 *intents* para cada tipo de información que se debe almacenar y se han activado las entregas para todos ellos. Para el nombre y la edad existen entidades definidas por el sistema que se pueden utilizar para obtener estos parámetros. La edad de diagnosis de la DM1 también se puede guardar como un parámetro distinto del mismo tipo de entidad que una edad normal. Sin embargo, se deben crear entidades nuevas para el tipo de diabetes y para el tratamiento. La entidad de tipo de DM solo tiene dos valores posibles: tipo 1 y tipo 2. En caso de que el usuario diga un tipo de DM distinta, el agente preguntará a qué tipo se asemeja más. Finalmente, para el tratamiento de la DM se han definido 4 posibilidades: insulina inyectada, una bomba de insulina, pastillas de metformina o una dieta sana.

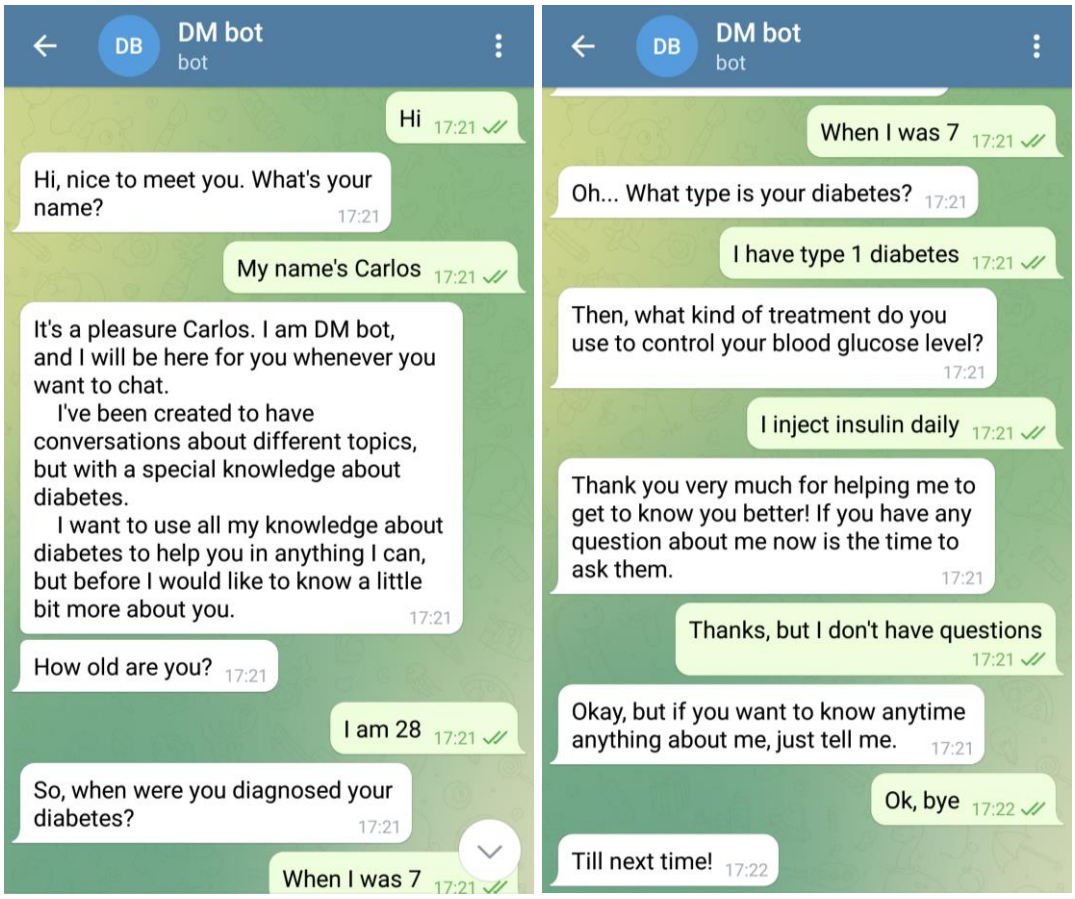

Figura 10: Ejemplo de conversación de introducción con el agente.

En el archivo index.js del se ha creado una función para cada *intent* de la conversación de introducción y se han conectado a los *intent* mediante los intentMap. Todas las funciones creadas tienen una estructura similar: primero se extrae la información del parámetro definido en el *intent*, después se almacena esa información en la base de datos y por último se escribe la respuesta que el agente dará al usuario. Para el almacenamiento de la información en la base de datos se utiliza la función saveToDB que se muestra en el Código 5.

```
async function saveToDB(document, key, value){
  let userId = agent.originalRequest.payload.data.from.id;
  const dbDocument = db.collection(""+userId).doc(""+document);
  await dbDocument.set({
    [key]:value
  }, { merge: true });
}
```

Código 5: Función para almacenar información obtenida por el agente en la base de datos.

Esta función tiene tres argumentos: el nombre del documento en el cual almacenar la información, la clave y el valor a guardar. Toda la información recopilada durante la conversación de introducción se guarda en un documento llamado Basic Info, que se crea

dentro de una colección cuyo nombre es la ID de Telegram del usuario. Esta ID se extrae a partir de los datos de carga útil, que es un tipo de información que envía Telegram a Dialogflow junto con el mensaje. De esta forma, la organización de la base de datos consistirá en una colección para cada usuario (nombrada con su ID de Telegram) que dentro contendrá inicialmente un documento con información básica. La forma de almacenar esta información en Firestore se muestra en la Figura 11.

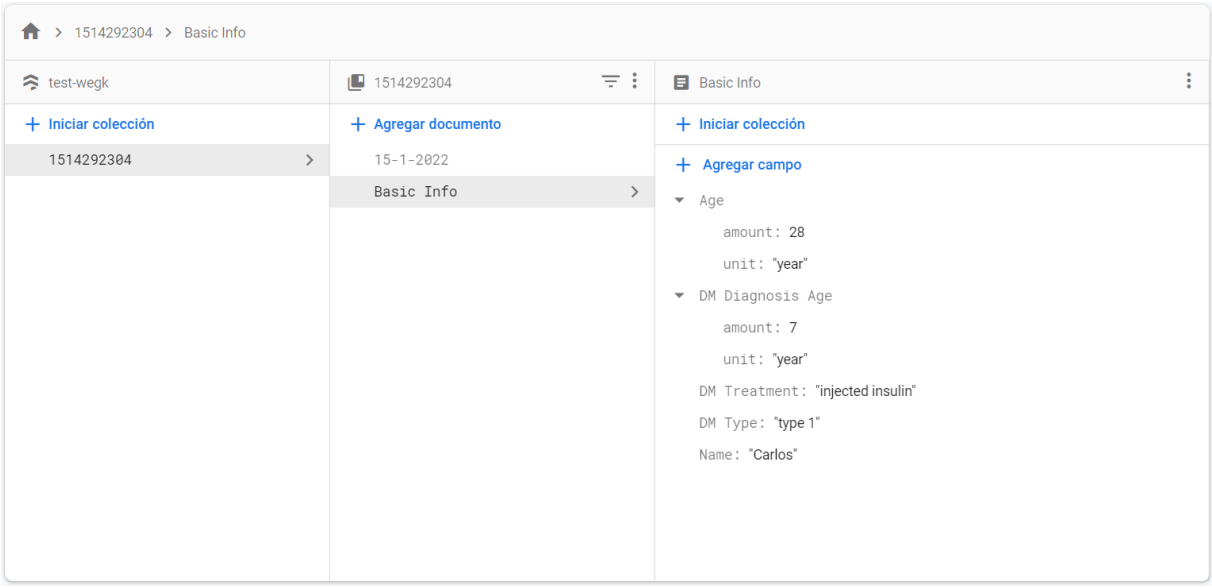

Figura 11: Almacenamiento de los datos de una conversación de introducción en Firestore.

5.6 Creación de *intents* con información de DM

Después de una primera conversación con un usuario para recopilar la información básica, el agente debe mantener conversaciones cuotidianas con el usuario. En ellas el agente pregunta por el transcurso del día y las actividades realizadas, además de la regulación de la DM. Para ello, se han creado algunos *intents* que contienen información relevante acerca del nivel de glucosa en sangre, la dosis de insulina administrada, si se ha realizado ejercicio físico, la ingesta de comida o si se da una situación que genere estrés. Todos estos *intents* tienen activadas las entregas para poder recoger toda la información útil y almacenarla en Firestore. La forma de almacenar la información en la base de datos Firestore que tienen estas funciones se explica a continuación.

Inicialmente se extrae la información de la fecha y hora de los datos de carga útil de Telegram. Esta fecha y hora está en formato Unix, y por tanto hay que transformarla a un formato de fecha más comúnmente utilizado. Esto se realiza mediante una función denominada UnixToDate, que recibe una marca de tiempo en formato Unix y lo convierte en una fecha en formato DD-MM-AAAA. Esta función se puede observar en el Código 6. Con la fecha en un

formato correcto, se comprueba si dentro de la colección de Firestore existe un documento con la fecha de ese día. En caso afirmativo, se añade el evento con toda la información relacionada con la DM dentro del documento de ese día. Si no existe el documento correspondiente al día, se crea y posteriormente se añade toda la información correspondiente.

```
function UnixToDate(timestamp) {
  const milliseconds = timestamp * 1000;
  const dateObject = new Date(milliseconds);
  return (dateObject.getDate()+"-"+(dateObject.getMonth()+1)+"-"+da-
teObject.getFullYear());
}
```

Código 6: Función para transformar un instante de tiempo en formato Unix a una fecha en formato DD-MM-AAAA.

Existen respuestas predefinidas para cada uno de estos *intents*, pero el objetivo es que las respuestas a estos *intents* las haga un modelo de generación de texto entrenado con conversaciones sobre diabetes. Esto hace que el agente sea menos repetitivo y que cada conversación se única y distinta cada vez.

5.6.1 Nivel de glucosa en sangre

Inicialmente se ha creado un *intent* que detecta cuando un mensaje contiene información relevante acerca del nivel de glucosa en sangre. En éste se puede obtener información de tres parámetros: un estado de glucosa, un valor de glucosa en sangre y el instante de tiempo en el que se ha detectado. Para detectar el estado de la glucosa (bajo, bien o alto) se ha creado una nueva entidad denominada estado de glucosa, mientras que el valor de la glucosa en sangre se recoge en un parámetro que corresponde a una entidad definida por el sistema, un número integro. Estos dos parámetros no son obligatorios, el mensaje del usuario puede contener uno de los dos parámetros o los dos. El último parámetro que se recoge es del tipo date-time definido también por el sistema, y este es un parámetro requerido. En caso de que el usuario diga un estado de glucosa sin especificar el momento, el agente preguntará para adquirir esta información.

Este *intent* tiene las entregas activas y una función asociada que recoge toda la información de los parámetros y crea un objeto de datos con el evento de glucosa y el instante de tiempo en que ha sucedido. Este objeto de datos se almacena en la base de datos utilizando la función saveToDB definida en la sección 5.5. En este caso, el documento donde se almacenan los datos tiene el nombre del día de la conversación, y los datos se guardan en una clave llamada “Evento de Glucosa x”. x corresponde al número de eventos de glucosa que se han

registrado ese día, y por tanto si durante una conversación un día se habla de tres niveles de glucosa distintos en tres momentos del día distintos, estos se almacenan como eventos distintos. El almacenamiento de la información en la base de datos se puede observar en la Figura 12. El formato de esta función en Node.js al igual que el formato de almacenamiento de la información en Firestore es muy similar al que se utiliza en todas las funciones asociadas a *intents* con información sobre la DM.

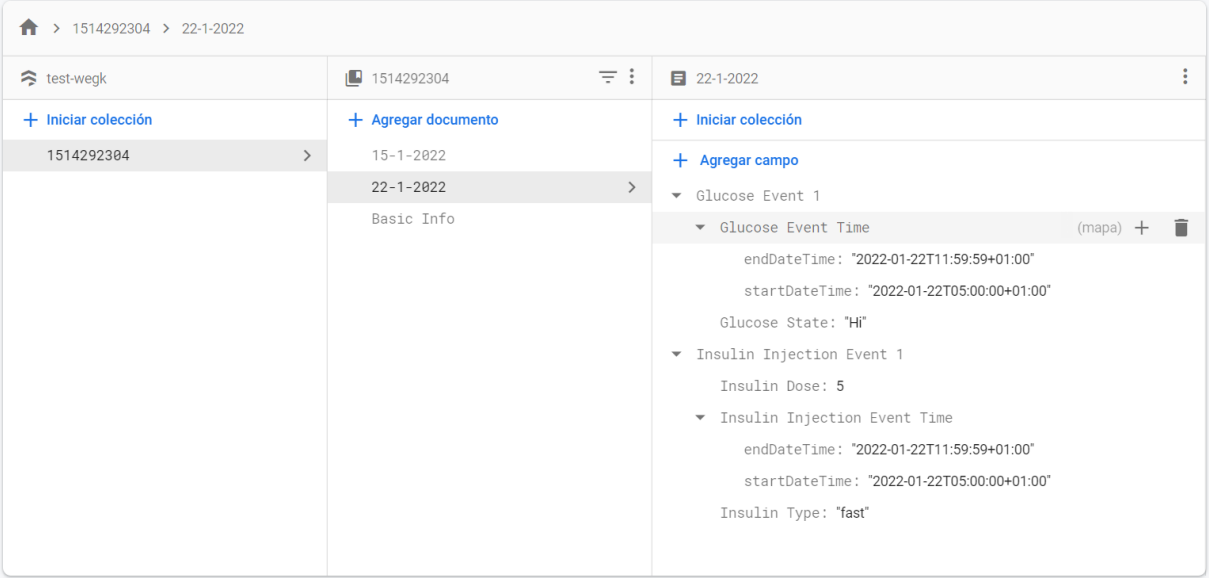

Figura 12: Ejemplo del almacenamiento en Firestore de la información sobre DM recopilada en una conversación diaria del usuario con el agente.

Una vez guardada la información, existe una respuesta predefinida en el agente con consejos para mejorar el nivel de glucosa dependiendo del estado que se haya detectado. Esto inicia una breve conversación en que se determina si el usuario ha seguido los consejos o no, y en caso afirmativo se podrá recopilar información de la acción realizada con los siguientes *intents*.

5.6.2 Dosis de insulina administrada

De forma similar al nivel de glucosa, la insulina administrada se detecta también con un único *intent*. En éste se detectan tres tipos de información: la dosis de insulina inyectada, el tipo de insulina y el instante de inyección de la dosis. En este caso todos estos parámetros son requeridos en el mensaje, y por tanto si el usuario no da información sobre alguno de ellos el agente hará preguntas hasta poder recoger toda la información necesaria.

Toda esta información se almacena también en la base de datos mediante la función saveToDB, almacenando la información de forma similar al ejemplo anterior. Ahora cada evento será de administración de insulina, y al igual que en el caso anterior se pueden

almacenar todos los que se hayan realizado cada día. Tras guardar la información sobre la dosis de insulina administrada, la respuesta predefinida del agente recordará la importancia de revisar la glucosa un tiempo después de la inyección para comprobar que la dosis ha sido adecuada.

5.6.3 Ejercicio físico

Los mensajes que contengan información sobre la actividad física se pueden clasificar en dos *intents* distintos, en función de si es ejercicio físico durante un periodo de tiempo o si es un partido o competición de un deporte. Cuando se trata de un ejercicio físico, se pueden recopilar tres tipos de información distinta: el tipo de ejercicio físico o deporte, la duración y cuándo ha sucedido. Para poder recoger esta información ha sido necesario crear una entidad nueva que contenga los deportes o tipos de ejercicio. Esta entidad consiste en una lista con distintos deportes o ejercicios, y sinónimos para algunos deportes para facilitar su identificación. El resto de los parámetros pertenecen a entidades del sistema, una es date-time comentada previamente y la otra expresa el tiempo de duración. De estos tres parámetros que se recopilan, tanto el tipo de deporte como la duración son obligatorios y el agente hace preguntas en caso de que el mensaje no aporte información sobre ellos. Por el contrario, el momento del día en que ha sucedido no es un parámetro requerido y por tanto se almacenará únicamente si se ha detectado dentro del mensaje.

Cuando el *intent* que se detecta corresponde a un partido o competición de algún deporte, a diferencia del *intent* anterior no se recopila información sobre la duración del partido. Por tanto, únicamente se recoge la información del momento en que se juega y el tipo de deporte. En este caso ambos parámetros son requeridos, y al igual que en casos anteriores el agente pregunta para recopilar la información que falte en el mensaje.

El almacenamiento de esta información en la base de datos se realiza de forma similar al resto de casos. Pese a que sea menos frecuente que estos eventos sucedan más de una vez al día, sigue existiendo la posibilidad y por ello se almacena la información en forma de eventos. La respuesta predefinida del agente en este caso varía en función de si se ha realizado ejercicio físico o si se ha jugado un partido. En este último el agente muestra interés por la competición y mantiene una conversación más larga con el usuario.

5.6.4 Ingesta de comida

Para detectar la ingesta de comida se ha definido únicamente un *intent*, que captura toda la información relevante referente a la comida. Este *intent* puede llegar a recoger cinco tipos de información distinta que se almacena en cinco parámetros. La información sobre el tipo de

comida se recoge en dos parámetros distintos, diferenciados por el nivel de carbohidratos de ésta. Para ello, se han definido dos nuevas entidades para estos dos tipos de comida relevantes en la DM: una que corresponde a comida con un nivel de carbohidratos bajo y otra con nivel de carbohidratos alto. Otro de los parámetros extraídos hace referencia al momento en que se ha ingerido comida, que corresponde al tipo de entidad date-time comentado anteriormente. Los últimos dos parámetros hacen referencia a la cantidad de comida, uno de ellos es el peso de la comida ingerida y el otro el número de un tipo de alimento que se ha comido. Estos dos últimos también pertenecen a entidades definidas por el sistema. De estos cinco parámetros, únicamente el momento de la ingesta es un parámetro requerido y el resto son opcionales. El tipo de comida siempre estará presente en el mensaje, pero puede ser de uno de los dos tipos o de ambos.

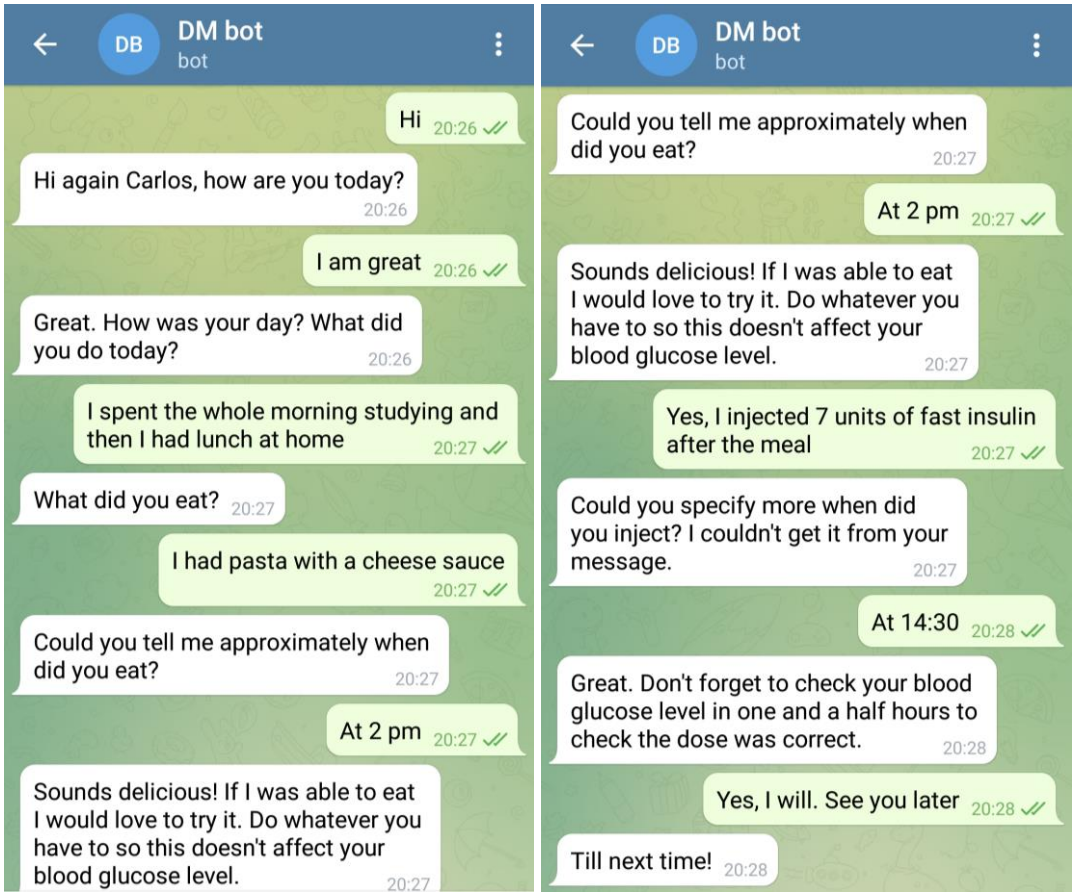

Figura 13: Ejemplo de conversación diaria de un usuario con el agente.

Este *intent* también tiene definida una función para almacenar toda la información proveniente del mensaje en la base de datos. En esta función se comprueban todos los parámetros que se han recibido en el mensaje del usuario, se crea un objeto de datos y se guarda en Firestore como “Evento de ingesta de comida” con el número adecuado. Finalmente, la respuesta predefinida del agente consiste en un breve recordatorio para un buen control de la glucosa en sangre. En la Figura 13 se observa un ejemplo de una conversación de un usuario con el

agente en la cual se activan *intents* con información sobre la DM descritos previamente y el agente envía respuestas predefinidas.

5.6.5 Situación de estrés

Dado que una situación de estrés puede afectar a los niveles de glucosa de un enfermo de DM, se ha decidido almacenar también esta información de una forma más simple. Para ello se ha definido un único *intent* en el cual se recoge información acerca de una fecha o momento en un único parámetro de tipo date-time. Este parámetro es requerido por el agente, ya que es la única información extraída del mensaje del usuario.

En lo referente a la entrega en este *intent*, únicamente se almacena el instante de tiempo en que sucede el evento que está causando estrés al usuario con la clave “Fecha del Evento de Estrés” y un número. A diferencia de los casos anteriores, el número de evento no se reinicia cada día, sino que es un número incremental. Después de guardar la información utilizando la función saveToDB, el agente sigue la conversación referente al evento estresante.

5.7 Implementación y conexión del modelo de GLN al agente conversacional

Hasta este punto se ha creado un agente capaz de mantener conversaciones acerca de la DM con un usuario y almacenar la información sobre la DM en una base de datos. Por tanto, las respuestas de este agente funcionan como las de un agente conversacional basado en reglas, pero el objetivo era crear un agente capaz de realizar respuestas dinámicas y distintas en cada conversación. Para ello, ha sido necesario crear un modelo de GLN que emplea las técnicas estudiadas en la sección 2.2.2 y que es capaz de elaborar una respuesta a los mensajes con información de DM de un usuario. Así, las respuestas generadas por este modelo se envían cuando el agente hace coincidir un mensaje del usuario con alguno de los *intents* definidos en la sección 5.6.

5.7.1 Entrenamiento y testeo del modelo de generación de lenguaje

El primer paso en el proceso de cambio de respuestas estáticas a respuestas dinámicas es el entrenamiento de un modelo capaz de generar texto de diálogos. Este modelo no se ha creado de cero, sino que se ha utilizado un modelo previamente entrenado con una cantidad masiva de datos. El modelo que se ha utilizado se denomina GPT-2 (*Generative Pre-trained Transformer 2*) (Radford, Wu, Child, Luan, & Amodei, 2019) y es un modelo de PLN basado en transformadores que se ha entrenado con bases de datos de textos masivas. Esto le

permite generar muestras de texto de alta calidad, apenas diferenciables de las que podría realizar un ser humano. El objetivo principal de este modelo es predecir la siguiente palabra dadas todas las palabras previas en un texto.

Para trabajar con este modelo no se ha importado directamente GPT-2, sino que se ha utilizado el paquete gpt-2-simple (Woolf, 2021). Este paquete de Python, además de las *scripts* existentes de reentrenamiento y generación de texto del modelo GPT-2, contiene un conjunto de funciones que simplifican la generación de texto. También tiene un notebook de Google Colab ya configurado para la descarga, reentrenamiento de modelos y generación de texto utilizando la GPU de los servidores de Google. Para el entrenamiento se recomienda utilizar siempre la GPU debido a que acelera el proceso en gran medida, aunque es posible utilizar la CPU también.

Las funciones de este paquete que principalmente se han utilizado son cuatro, y se explican a continuación. Primero, una función para descargar el modelo deseado de entre cuatro modelos disponibles con 124, 355, 774 y 1558 millones de parámetros. Segundo, una función para cargar un *checkpoint* de entrenamiento, que se guarda automáticamente cuando se entrena el modelo. Tercero, una función para reentrenar el modelo elegido denominada *finetunning*, que se puede utilizar sobre modelos ya reentrenados o limpios de GPT-2. Finalmente, una función con la que generar texto que permite el uso de un prefijo o, en este caso, el mensaje al cual se pretende responder.

El modelo GPT-2 recién descargado es capaz de generar texto gramatical, ortográfica y semánticamente correcto, pero el tipo de texto creado no es el deseado para la aplicación de este proyecto. Para que pueda generar texto de un estilo concreto como diálogos o conversaciones de chat, es necesario reentrenarlo utilizando bases de datos que contengan texto de ese mismo estilo. Este proceso se denomina *finetunning*, y se puede llevar a cabo para que el modelo sea capaz de generar diferentes tipos de texto o incluso texto en diferentes idiomas. En este proyecto se ha realizado una búsqueda de bases de datos de conversaciones para entrenar chatbots en Kaggle y finalmente se ha seleccionado una de más de 184000 mensajes: *Chatbot Dataset Topical Chat* (AS, 2021). Además de ésta, también se ha creado y utilizado la base de datos con información sobre la DM que se ha comentado en la sección 2.3.5. Esta base de datos creada manualmente se ha subido a Kaggle y se puede observar en el Anexo II.

De los cuatro modelos GPT-2 disponibles se ha decidido entrenar únicamente dos, uno de 124 millones y el otro de 355 millones de parámetros. Se han seleccionado los modelos más pequeños porque los más grandes pueden tardar demasiado tiempo en generar una respuesta, cosa que empeora la calidad de la comunicación con el usuario. Ambos modelos

se han entrenado siguiendo el mismo proceso y con las mismas configuraciones, pero en entornos distintos. Primero se ha hecho *finetunning* de 1000 pasos con la base de datos de conversaciones de actualidad obtenida de Kaggle. Posteriormente se ha entrenado únicamente 100 pasos con la base de datos de conversaciones con información de DM y se ha aumentado el *learning rate* a 1e-5 debido a que la base de datos era muy pequeña. El modelo de 124 se ha entrenado en el Notebook de Google Colab del Anexo IV siguiendo este proceso, pese a que únicamente se observa el último entrenamiento. Después se ha descargado el *checkpoint* del entrenamiento para poder ejecutarlo en local. El modelo de 355 se ha entrenado en un Notebook de Python en local debido a que no era posible el entrenamiento en Google Colab, y todo este proceso se puede observar en el Notebook del Anexo III. Una vez entrenados ambos modelos se han probado poniendo como prefijo un mensaje con información sobre la DM que podría enviar un usuario y comprobando que se reciben respuestas coherentes. La Figura 14 muestra un ejemplo de respuesta generada a una simulación del mismo mensaje de un usuario por el modelo de 124 millones de parámetros y por el modelo 355 millones de parámetros.

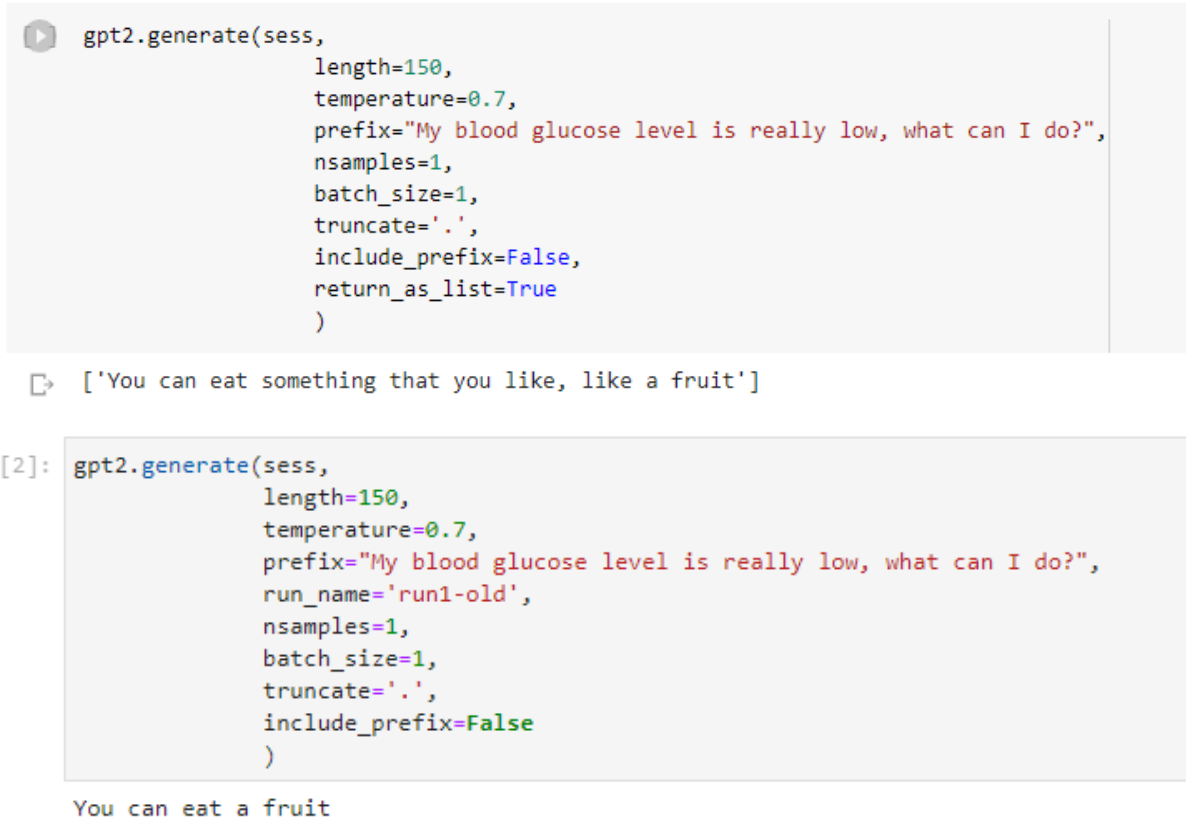

Figura 14: Generación de una respuesta al mismo mensaje utilizando el modelo de 124 millones de parámetros (arriba) y el de 355 millones de parámetros (abajo).

5.7.2 Uso del modelo como una aplicación web

Dado que no es posible hacer llamadas a un Notebook de Python en local desde un entorno distinto, ha sido necesario crear una *script* de Python y convertirla en una aplicación web. Esto se ha realizado utilizando la aplicación Flask (Pallets, 2010), que es un *framework* que permite crear aplicaciones web de forma simple. Al importar Flask en la *script* creada, únicamente ha sido necesario asignarle un puerto local y definirla siguiendo la documentación. Para que el modelo se pueda ejecutar en local utilizando una script de Python en lugar del Notebook, ha sido necesario instalar las versiones adecuadas de tensorflow-gpu, CUDA y cuDNN. Tensorflow es una librería de código abierto de las más utilizadas en la actualidad para aprendizaje automático, y tensorflow-gpu permite el desarrollo de este tipo de computación en esta plataforma. CUDA (*Compute Unified Device Architecture*) es una plataforma de computación en paralelo y cuDNN (*CUDA Deep Neural Network*) es un paquete de NVIDIA. Gracias a estos es posible utilizar una GPU de NVIDIA para el procesamiento de modelos. La información de las versiones necesarias se ha extraído de la documentación oficial de Tensorflow, ya que las últimas versiones de cada librería o plataforma daban error al trabajar con el modelo.

Una vez configurado el *script* de Python, ha sido necesario crear otro *script* que contenga dos funciones para modificar el formato del texto de entrada y de salida del modelo. Ambos *scripts* se observan en el Anexo III con los nombres Model.py y message_formatter.py. Los mensajes introducidos deben contener un signo de puntuación para finalizar, y en caso de que no tengan, es necesario añadir uno. Esto es debido a que normalmente no se utilizan signos de puntuación en las conversaciones de texto escrito, pero GPT-2 fue creado para generar texto ortográficamente correcto. Al ser un modelo de generación de texto, a veces no es capaz de devolver únicamente un turno de conversación de un diálogo y lo continúa por sí solo, cosa que no interesa para la aplicación. Por tanto, se ha restringido la respuesta generada a un solo mensaje, que puede contener más de una oración.

Finalmente, se ha utilizado la herramienta Postman (Postman, 2022) para comprobar que es posible realizar llamadas a la aplicación web donde está el modelo y recibir respuestas adecuadas. Siempre se envía información en formato JSON, el modelo extrae el mensaje de ese JSON con información, lo procesa y devuelve un nuevo JSON con la respuesta. Utilizando la herramienta Postman se envía un mensaje al puerto definido por Flask del localhost.

5.7.3 Llamada al modelo desde index.js de Dialogflow

En este punto el modelo se encuentra en una aplicación web asignada a un puerto local, pero para poder acceder desde los servidores de Firebase ha sido necesario utilizar la aplicación ngrok (ngrok, 2022). Esta aplicación crea una URL dinámica que apunta a un servicio web en ejecución dentro de nuestra máquina local, haciendo público un puerto de nuestro *localhost* y permitiendo que se pueda acceder desde cualquier entorno. Una vez descargada y ejecutada la aplicación, se ha comprobado mediante la herramienta Postman que enviando un mensaje en formato JSON a la URL dinámica creada mediante ngrok también se obtiene una respuesta del modelo.

```
function callModel(question, callback) {
  const promise = new Promise((resolve) => {
    let data = {message:question};

    console.log(`callModel: ${JSON.stringify(data)}`);

    try {
      rp({
        method: "POST",
        uri: "https://e8bd-46-24-247-212.ngrok.io/api",
        body: data,
        headers: {
          "Content-Type": "application/json"
        },
        json: true
      }).then(answer => {
        console.log(`API call answer: ${JSON.stringify(answer)}`);
        callback(answer.response);
        resolve();
      }).catch(function (err) {
        console.log(`API call error: ${err}`);
        resolve();
      });
    } catch (error) {
      console.log(`API error: ${error}`);
      resolve();
    }
  });

  return promise;
}
```

Código 7: Función que llama al modelo de GLN y devuelve una respuesta dinámica.

El último paso para que el agente pueda generar respuestas dinámicas es hacer la llamada a la URL dinámica desde el archivo index.js. Esto se ha realizado mediante la función callModel, que se puede observar en el Código 7. Ésta toma como argumentos el mensaje enviado por el usuario y la función a ejecutar cuando se obtenga la respuesta del modelo. Inicialmente se configura el mensaje en un formato JSON y utilizando el paquete *request-promise* se envía el JSON de datos a la URL definida utilizando el método POST. Cuando se recibe una respuesta del modelo se ejecuta la función *callback* introducida como argumento y finalmente la función callModel devuelve la promesa cumplida para impedir que la función pueda quedar en ejecución.

Una vez configurado todo, se ha desplegado y probado, y pese a que la información se envía y se recibe correctamente, Dialogflow tiene definido un *timeout* para entregar una respuesta cuando un *intent* utiliza entregas. Si la respuesta no llega en menos de 5 segundos, Dialogflow envía la respuesta predefinida añadida en la configuración del *intent*. El modelo más pequeño que se ha entrenado tarda de 6 a 8 segundos en generar una respuesta, mientras que el modelo más grande tarda entre 20 y 25 segundos. A esto hay que sumarle el tiempo que tarda el agente en enviar el mensaje al modelo y el tiempo de recepción del mismo. Por tanto, es imposible que ninguna de las respuestas generadas por el modelo llegue a tiempo para salir por el chat de Telegram y siempre se mostrarán las predefinidas. Sin embargo, las respuestas generadas por el modelo se pueden observar en el registro de Firebase tal y como muestra la Figura 15.

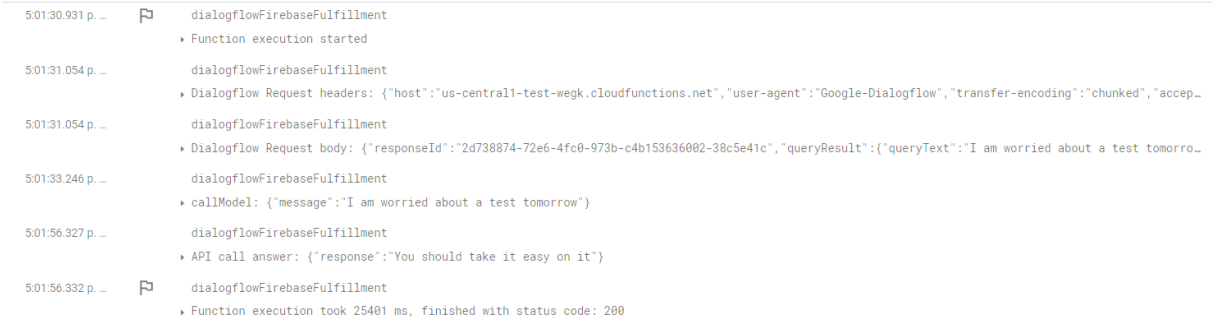

Figura 15: Registro de Firebase functions donde se observa el mensaje que se envía al modelo mediante la función callModel, la respuesta recibida y el tiempo de ejecución de todo el proceso.

Tras una búsqueda para ampliar este *timeout* de 5 segundos establecido por Dialogflow sólo se han encontrado dos opciones. La primera consiste en crear el agente utilizando Dialogflow CX en lugar de utilizar Dialogflow ES, ya que en esa edición es posible ampliar el *timeout* de un *webhook* hasta 30 segundos. Dialogflow CX está pensado para crear agentes más grandes con usos más complejos, y por tanto está enfocado a empresas de un cierto tamaño con un departamento centrado en esto. Sin embargo, Dialogflow ES está enfocado a aplicaciones más simples hechas por pequeñas empresas o desarrolladores independientes. Pese a que

ambas ediciones pertenecen al mismo *framework*, cada una está construida en un modelo de datos completamente distinto a la otra, cosa que dificulta en gran medida una migración de un proyecto entre ambas. Por tanto, sería necesario volver a crear el agente desde cero en esta nueva edición de Dialogflow. Esto se ha desestimado debido a que es un proceso laborioso y no se goza del tiempo suficiente para llevarlo a cabo.

La segunda opción es extender el *timeout* haciendo una llamada un a evento de seguimiento, que a su vez puede realizar otra llamada a otro evento de seguimiento llegando a conectar tres *intents* mediante estos dos eventos, aumentando el tiempo a 15 segundos. Tras haber creado los *intents* y eventos de seguimiento, y haber configurado las funciones de entrega del archivo index.js, no ha sido posible lograr que se llame a los eventos de seguimiento. Esto se debe a que para que la función asociada a un *intent* pueda realizar correctamente la llamada al modelo, esta función debe devolver la llamada a callModel. Si la función asociada al *intent* no devuelve la llamada a callModel, esta función no espera a que se envíe la información y se reciba la respuesta, sino que directamente da la respuesta predefinida en Dialogflow y devuelve un error en el registro. Para poder establecer eventos de seguimiento la llamada al modelo debería realizarse antes de la llamada al evento de seguimiento, pero esto nunca sucederá porque callModel debe ser devuelta por la función asociada al *intent* para funcionar correctamente. La única solución posible sería almacenar la respuesta del modelo en otra base de datos y que los eventos de seguimiento comprueben si esta base de datos ha sido escrita, pero por limitaciones de tiempo y dificultad se ha desestimado en este proyecto.

6 Evaluación

Dentro de los objetivos de este proyecto había la evaluación de la funcionalidad y del uso del agente conversacional. Dado que no ha sido posible cumplir con el objetivo inicial del proyecto debido a una restricción del *framework* de creación de agentes conversacionales, la evaluación funcional se ha centrado en el modelo de IA y por tanto ha sido necesario realizar dos evaluaciones distintas. Por un lado, se ha evaluado el uso del agente conversacional mediante uno de los cuestionarios estandarizados más utilizado, la *System Usability Scale* (SUS) (Brooke, 1996). Por otro lado, se ha realizado una evaluación de la coherencia de las respuestas creadas por el modelo de generación de texto a los mensajes con información sobre la DM enviados por el usuario.

6.1 Evaluación del uso del agente conversacional

Siguiendo la metodología iterativa e incremental establecida para este proyecto, esta evaluación corresponde a la primera iteración. Esto implica que el agente se encuentra en una etapa inicial y en un estado similar a un prototipo, al cual falta por añadir funcionalidades. Por esta razón, por el desarrollo de la herramienta en inglés y por problemas de tiempo se ha testeado este agente conversacional con respuestas estáticas con cuatro usuarios diabéticos. Todos estos usuarios sufren DM tipo 1 desde hace más de 10 años.

6.1.1 System Usability Scale

El SUS es una herramienta rápida y confiable para evaluar la usabilidad de prácticamente cualquier tipo de sistema computacional, producto o servicio. En la actualidad se considera un estándar de este tipo de evaluaciones, con más de 12000 citaciones (tal y como se observa en Google Scholar). Esto es debido a que se puede administrar de forma sencilla a los usuarios y no requiere de un número de usuarios elevado para generar unos resultados confiables.

Esta herramienta consiste en un cuestionario de 10 preguntas con 5 opciones de respuesta numeradas para cada pregunta, desde totalmente en desacuerdo (uno) a totalmente de acuerdo (cinco). Este cuestionario se puede observar en la Figura 16. Las preguntas de numeración impar tienen un tono positivo y, por tanto, su contribución a la puntuación final corresponde a su valor en la escala de respuesta menos 1. Las preguntas de numeración par, por el contrario, tienen un tono negativo y su contribución a la puntuación es 5 menos el valor en la escala de respuestas. Finalmente se suman todas las puntuaciones de las preguntas y se multiplican por 2.5 para obtener la puntuación SUS.

El rango de la puntuación SUS va de 0 a 100, pero no se corresponde con un porcentaje. Por eso, una puntuación SUS por debajo de 68 se considera por debajo de la media y por tanto es mejorable. Una puntuación superior a 80 se consideraría muy buena, mientras que una puntuación menor a 50 implicaría que la aplicación tiene serios problemas de usabilidad.

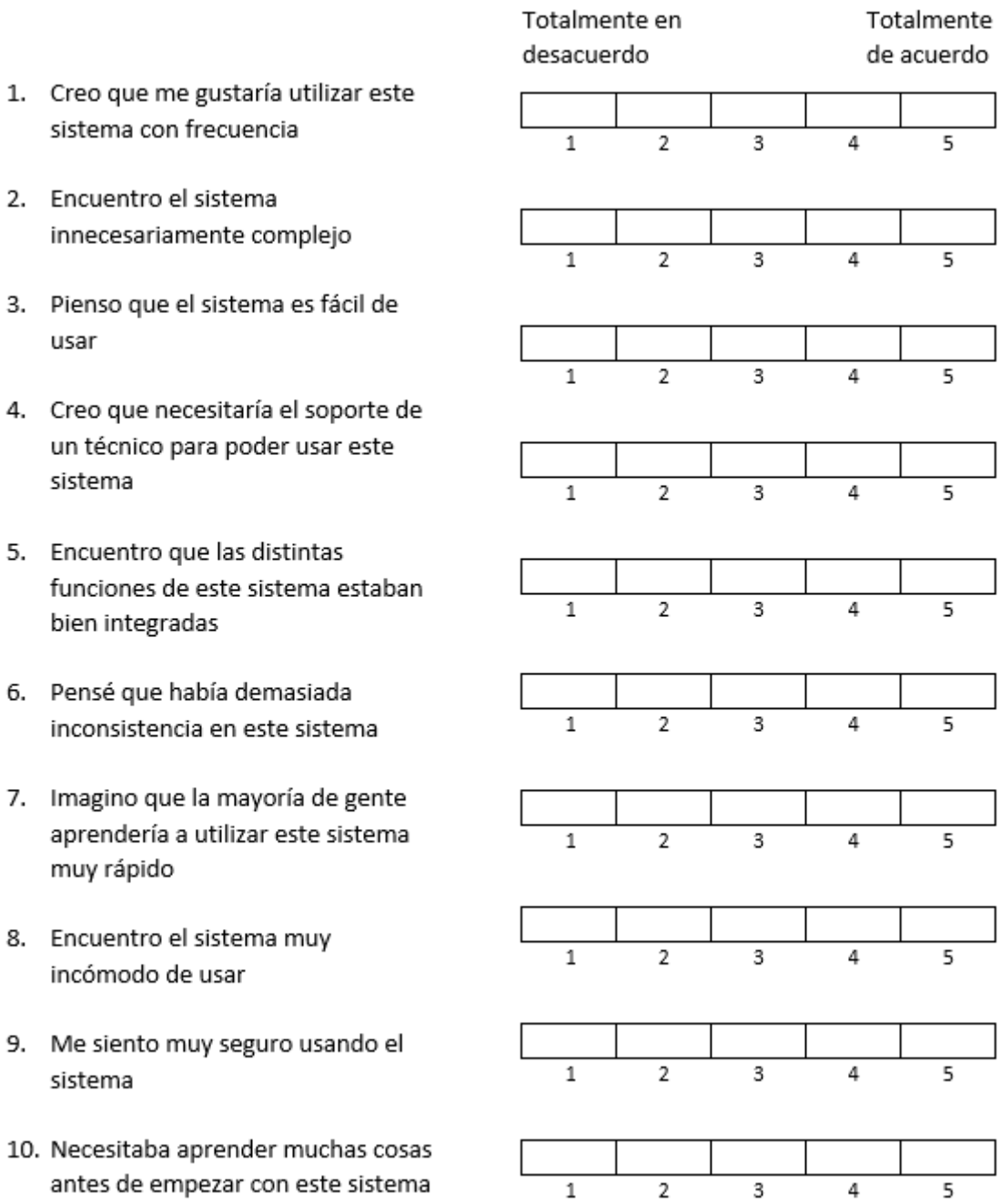

Figura 16: Cuestionario SUS traducido al español.

6.1.2 Resultados de los usuarios a la SUS

La valoración de la primera usuaria al sistema mediante la SUS ha sido de 75. Pese a que es una valoración por encima de la media, implica que hay todavía una mejora posible en la usabilidad de la herramienta. Tras analizar el cuestionario, las puntuaciones más bajas de este usuario indican que hay inconsistencia en el sistema y que no aporta seguridad al usuario. La inconsistencia en el sistema es debida a que las conversaciones con este usuario han tenido algún mensaje que no ha activado ningún *intent*. Esto puede tener dos posibles causas: que existen pocos *intents* configurados o que los *intents* no tienen suficientes frases de entrenamiento para una activación correcta. En ambos casos hay que mejorar la configuración del agente para que sea capaz de mantener conversaciones más variadas y correctas. En referencia a la seguridad usando el sistema, es comprensible que no se valore alto debido a que una persona no se siente segura delegando decisiones importantes sobre su salud a un agente conversacional. Finalmente, la usuaria ha indicado que la idea detrás del proyecto es buena y que si el agente llegase a un estado más evolucionado podría ayudar a muchos diabéticos.

El segundo usuario que ha valorado el sistema es un varón de 33 años y la puntuación de la herramienta ha sido 80, mejor que en el caso anterior. Las valoraciones más bajas de este usuario han sido la frecuencia de uso del agente y, al igual que la anterior usuaria, la seguridad que aporta el uso del sistema. Este usuario ha indicado que es una buena herramienta para poder hablar de diabetes, pero que las conversaciones se vuelven repetitivas cuanto más se usa el agente y eso conlleva una disminución del uso de la herramienta a medida que pasa el tiempo.

La tercera usuaria que ha utilizado el sistema es una mujer diabética de 61 años, y la valoración que ha dado de la herramienta ha alcanzado una puntuación SUS de 95, la más alta de todos los usuarios. Esta usuaria ha remarcado el gran potencial de la aplicación y la utilidad de los recordatorios que ayudan a una mejor gestión de la DM. Las valoraciones más bajas de esta usuaria han sido las mismas que las del usuario anterior, pero en el caso de la frecuencia de uso la causa es distinta. Una mayor edad de esta usuaria implica que no está tan acostumbrada a usar este tipo de tecnología, y por tanto la intención de uso del agente es más baja. A pesar de esto, la usuaria ha indicado que esto podría cambiar en el futuro si el uso del agente se integrara en su vida diaria.

La última valoración que ha recibido el sistema ha acumulado una puntuación SUS de 92,5. Esta valoración ha sido realizada por una usuaria de 28 años. Las valoraciones más bajas de esta usuaria han sido las mismas que las de la primera usuaria, pese a que la puntuación

global ha sido mayor. De forma similar a la primera usuaria, estas puntuaciones bajas son debidas a que el agente no ha detectado algunos *intents* de glucosa correctamente. Esto es normal al ser una primera versión o prototipo, pero a partir de esta evaluación se completarán las frases de entrenamiento del agente para reducir el número de mensajes de usuarios que el agente no es capaz de asociar a *intents*.

La media de las puntuaciones SUS de todos los usuarios ha sido 85,5, cosa que demuestra la usabilidad de la herramienta. Si se analizan todas las valoraciones de forma global, la puntuación más baja obtenida hace referencia al sentimiento de seguridad utilizando la herramienta. Esto es lógico, debido a que la DM es una enfermedad sensible en la que una mala dosis de insulina puede causar malestar corporal, mareos o hasta desmayos. Pese a que el agente nunca llega a dar indicaciones de dosis de insulina o de ingesta de carbohidratos, un diabético siempre se fiará más de un humano que de un agente conversacional.

6.2 Evaluación del modelo de generación de texto

De los dos modelos que inicialmente se entrenaron para hablar de diabetes, finalmente se decidió utilizar el de 124 millones de parámetros. Esto es debido a que el tiempo de generación de respuesta de este modelo es menor al del modelo con más parámetros, y en una conversación con usuarios un punto importante en la satisfacción es la velocidad de recibir el mensaje en la conversación.

Debido a los problemas funcionales del modelo con el *timeout* de Dialogflow, se ha decidido deshabilitar las llamadas al modelo en el index.js para que los mensajes de usuario que activan *intents* con información de DM no tarden más tiempo del que deberían en enviar la respuesta predefinida. Por tanto, el modelo se ha evaluado en local con ejemplos de mensajes de usuario extraídos del registro de conversaciones del agente.

Tras una búsqueda de métodos de evaluación de agentes conversacionales que emplean GLN, no se ha encontrado ninguna metodología estandarizada. Únicamente se han encontrado métodos de evaluación de modelos de generación de texto que se clasifican en tres categorías (Celikyilmaz, Clark, & Gao, 2020): evaluación centrada en el ser humano, métricas automáticas sin entrenar y métricas de aprendizaje automático. Las métricas automáticas sin entrenar comparan textos generados por humanos y por modelos basados en los mismos datos de entrada. Esta comparación no precisa aprendizaje automático, sino que se superponen directamente ambos textos y se analiza la diferencia en los *strings*, en el contenido o la diferencia léxica de forma automática. A diferencia de estas, las métricas de

aprendizaje automático usan modelos para realizar este análisis de diferencias. En estos dos métodos de evaluación se requiere la generación de grandes cantidades de texto para la evaluación y el objetivo del modelo de este proyecto es la generación de mensajes de chat breves. Por tanto, no se ha considerado adecuado utilizar estos dos métodos y se ha decidido hacer una evaluación personalizada centrada en el ser humano, en la cual se ha juzgado la validez de las respuestas creadas por el modelo.

```
gpt2.generate(sess,
                  length=150,
                  temperature=0.7,
                  model_name='124M',
                  prefix="I think my glucose is very low, I need to do something.",
                  nsamples=1,
                  batch_size=1,
                  truncate='.',
                  include_prefix=False,
                  return_as_list=True
                  )
```

```
["Have you tried to eat something, such as a banana?"]
```

```
gpt2.generate(sess,
                  length=150,
                  temperature=0.7,
                  model_name='124M',
                  prefix="I think my glucose is very low, I need to do something.",
                  nsamples=1,
                  batch_size=1,
                  truncate='.',
                  include_prefix=False,
                  return_as_list=True
                  )
```

```
["It will probably be low, because you'll want to do something more difficult"]
```

```
gpt2.generate(sess,
                  length=150,
                  temperature=0.7,
                  model_name='124M',
                  prefix="I think my glucose is very low, I need to do something.",
                  nsamples=1,
                  batch_size=1,
                  truncate='.',
                  include_prefix=False,
                  return_as_list=True
                  )
```

```
['Just drink some water']
```

Figura 17: Categorías de respuesta del modelo al mismo mensaje del usuario. Arriba se consideraría una respuesta correcta, en medio una respuesta incoherente y debajo una respuesta incorrecta.

Las respuestas generadas por el modelo se han clasificado en tres categorías en función de si eran correctas, incorrectas o incoherentes. Las respuestas se consideran incorrectas cuando tienen sentido lógico en la conversación, pero el contenido de las mismas no es

adecuado. La Figura 17 muestra un ejemplo de respuesta de cada tipo generada por el modelo para el mismo mensaje de entrada.

Los ejemplos de mensajes de usuario se han clasificado en cinco tipos distintos en función de su temática. Esta temática está alineada con los *intents* definidos en el apartado 5.6. De esta forma, se evalúan las respuestas a mensajes sobre niveles de glucosa del usuario, sobre una comida hecha por el usuario, acerca de una actividad física realizada por el usuario, sobre una dosis de insulina inyectada por el usuario y mensajes donde el usuario expresa una situación que le genera estrés.

La Tabla 1 muestra el número total de mensajes de cada tipo que se han probado en el modelo y la clasificación de cada respuesta generada en las categorías definidas previamente. A primera vista se observa que los tipos de mensaje que han recibido un número de respuestas correctas proporcionales mayor han sido los relacionados con ejercicio físico, comida y estrés. Estas respuestas no necesariamente contenían consejos o recordatorios para una mejor gestión de la diabetes, pero eran respuestas coherentes dentro de la conversación. Por el contrario, los mensajes a los que peores respuestas correctas se han generado han sido los que contenían información de niveles de glucosa y de dosis de insulina. Al analizar más a fondo el funcionamiento del modelo con mensajes de niveles de glucosa, se ha observado que el modelo generaba peores respuestas cuando el nivel de glucosa se indicaba numéricamente que cuando se hacía de forma cualitativa.

Tipos de mensaje	Respuestas correctas	Respuestas incorrectas	Respuestas incoherentes	Suma total de respuestas
Glucosa	19 (35,9%)	13 (24,5%)	21 (39,6%)	53
Insulina	10 (29,4%)	10 (29,4%)	14 (41,2%)	34
Ejercicio	16 (59,3%)	3 (11,1%)	8 (29,6%)	27
Comida	19 (51,4%)	10 (27%)	8 (21,6%)	37
Estrés	13 (50%)	3 (11,5%)	10 (38,5%)	26

Tabla 1: Resultados de la clasificación de las respuestas generadas por el modelo a los distintos tipos de mensaje de los usuarios.

La causa principal por la que el modelo genera mejores respuestas a *intents* de ejercicio físico, comida y estrés es debido a que estos mensajes no están ligados con la DM directamente, sino que pueden aparecer en una conversación casual con un agente conversacional. La primera base de datos de diálogos que se ha utilizado para entrenar el modelo contenía conversaciones casuales entre las cuales aparecían mensajes con este tipo de información. Al ser una base de datos mucho más grande que la segunda, que estaba centrada en la DM, ha permitido un mejor entrenamiento y por tanto una generación de respuestas mejor que

para la glucosa e insulina. Finalmente, la categoría de mensajes que ha obtenido el mayor número de respuestas correctas ha sido la que aporta información sobre el ejercicio físico, con un valor de 59,3% de respuestas correctas. Pese a que más de la mitad de respuestas generadas por el modelo son correctas, el valor obtenido está lejos de ser un valor óptimo.

De esta forma se puede concluir que el modelo necesita más entrenamiento con conversaciones casuales y especialmente con información de DM como los niveles de glucosa o las dosis de insulina. Estos resultados son coherentes teniendo en cuenta la metodología propuesta para este proyecto y el estado actual del mismo. El siguiente paso a la hora de mejorar este modelo es utilizar el histórico de conversaciones de los usuarios con el agente para volver a realizar una fase de entrenamiento. Por tanto, en este estado no sería adecuado utilizar las respuestas dinámicas en el agente debido a que para mensajes de glucosa y dosis de insulina el modelo tiene un bajo porcentaje de respuestas correctas y si un usuario siguiera una recomendación incorrecta podría ser perjudicial para su salud.

7 Conclusiones y trabajo futuro

7.1 Conclusiones

El objetivo principal de este proyecto era crear un agente conversacional para el almacenamiento de información de la diabetes mellitus (DM) tipo 1. Tras un amplio estudio de la bibliografía y de las herramientas disponibles, se ha creado un agente utilizando Dialogflow ES capaz de detectar información de diabetes en una conversación cuotidiana y almacenarla en la base de datos asociada a este *framework*, Firebase. No ha sido posible que este agente utilice la información sobre la DM almacenada durante una conversación en diálogos futuros, y por tanto queda pendiente para la siguiente etapa de desarrollo. También se ha creado una metodología funcional para implementar respuestas dinámicas en el agente conversacional mediante el entrenamiento de un modelo de generación de lenguaje natural (GLN), GPT-2. Esta metodología finalmente no ha funcionado siguiendo su propósito inicial debido a una limitación del *framework* que establecía un tiempo límite de 5 segundos para enviar una respuesta al usuario, mientras que las respuestas dinámicas generadas por el modelo tardaban como mínimo 8 segundos. Por esta razón ha sido necesario evaluar el agente conversacional y el modelo de GLN por separado, dividiendo de esta forma los objetivos de evaluar la funcionalidad y la usabilidad de la herramienta.

La usabilidad de la herramienta se ha evaluado mediante la *System Usability Scale* (SUS), un cuestionario comúnmente utilizado para evaluar el uso de cualquier tipo de sistema computacional. La puntuación media obtenida usando la SUS ha sido de 85,5, lo cual indica que la herramienta creada tiene buena usabilidad. La evaluación del modelo ha mostrado que cuando el agente recibe mensajes con información de comida, ejercicio físico o estrés, la respuesta dinámica generada por el modelo será correcta para la mitad o más de estos mensajes. Sin embargo, cuando el agente reciba mensajes con información de niveles de glucosa del usuario o dosis de insulina inyectada, una de cada tres respuestas generadas por el modelo será correcta. Esto demuestra que el modelo de generación de texto requiere más entrenamiento con conversaciones para ser capaz de generar más respuestas dinámicas correctas.

7.2 Líneas de trabajo futuro

Siguiendo con la metodología establecida al inicio del proyecto, el estado actual del proyecto se podría considerar como un prototipo. El siguiente paso en esta dirección sería diseñar el

agente en un *framework* capaz de ampliar el *timeout* para enviar una respuesta, lo cual permitiría añadir las respuestas dinámicas de forma totalmente funcional. También sería necesario reentrenar el modelo de generación de texto con históricos de conversaciones para que fuera capaz de generar respuestas dinámicas más coherentes con el hilo de la conversación. Finalmente, se debería ampliar el número de *intents* que recopilan información de la DM y configurarlos para almacenar la información en Firebase.

Otro aspecto que actualmente está poco desarrollado es el procesamiento del lenguaje natural en idiomas distintos al inglés. Pese a que en este proyecto se ha creado un agente bilingüe, el idioma principal del mismo ha sido el inglés. Sin embargo, se podrían utilizar un histórico de conversaciones en español con el agente para volver a entrenar el modelo GPT-2. De esta forma se generarían más bases de datos en español y más adelante se podría llegar a crear una herramienta similar a la desarrollada pero totalmente en español.

Además, en un futuro sería ideal que el agente fuese integrado con más plataformas de mensajería, para poder tener mayor acceso por parte de usuarios que no tengan Telegram. Una característica que también se debería implementar sería la capacidad por parte del agente de dar pequeños consejos relacionados con la gestión de la DM. De esta forma, si se detectase un episodio de glucosa baja en la conversación, el agente podría recomendar la ingesta de carbohidratos siguiendo lo estipulado oficialmente por la American Diabetes Association. La capacidad de realizar recomendaciones mayores y de más peso sería un proyecto futuro que continuaría el trabajo realizado en este.

8 Bibliografía

A. W. (2021). *Chatbot | Deep Learning | Amazon Lex*. Obtenido de https://aws.amazon.com/es/lex/

Adamopoulou, E., & Moussiades, L. (2020). Chatbots: History, technology, and applications. *Machine Learning with Applications*(2).

Alfian, G., Syafrudin, M., Anshari, M., Benes, F., Atmaji, F. T., Fahrurrozi, I., & Hidayatullah., A. F. (2020). Blood glucose prediction model for type 1 diabetes based on artificial neural network with time-domain features. *Biocybernetics and Biomedical Engineering, 40*(4), 1586-1599.

Al-Taee, M. A., Al-Nuaimy, W., Muhsin, Z. J., & Al-Ataby, A. (2017). Robot Assistant in Management of Diabetes in Children Based on the Internet of Things. *IEEE INTERNET OF THINGS JOURNAL*.

Amazon. (2019). *Amazon Alexa Official Site: What is Alexa?* Obtenido de https://developer.amazon.com/es-ES/alexa

American Diabetes Association. (2018). Good to Know: Factors Affecting Blood Glucose. En *Clinical Diabetes* (págs. 36(2): 202-202).

American Diabetes Association. (2021). 2. Classification and diagnosis of diabetes: Standards of Medical Care in Diabetes. En *Diabetes Care* (págs. 44(Suppl. 1):S15 - S33).

Apple. (s.f.). *Siri*. Obtenido de https://www.apple.com/siri/

AS, A. S. (Febrero de 2021). *Chatbot Dataset Topical Chat*. Obtenido de Kaggle: https://www.kaggle.com/arnavsharmaas/chatbot-dataset-topical-chat

Botsify. (2021). *Botsify - A Fully Managed Chatbot Platform To Build AI-Chatbot*. Obtenido de https://botsify.com/

Brooke, J. (1996). SUS - A quick and dirty usability scale. *Usability evaluation in industry*.

Brown, B., Nasirzada, N., & Benson, D. (10 de Septiembre de 2021). *Botkit: Building Blocks for Building Bots*. Obtenido de https://github.com/howdyai/botkit/blob/main/packages/docs/index.md

Celikyilmaz, A., Clark, E., & Gao, J. (2020). Evaluation of Text Generation: A Survey. *arXiv:2006.14799*.

Chaki, J., Ganesh, S. T., Cidham, S., & Theertan, S. A. (2020). Machine learning and artificial intelligence based Diabetes Mellitus detection and self-management: A systematic review. *Journal of King Saud University - Computer and Information Sciences*.

Chatfuel. (2021). *Chatfuel: world-leading chatbot solution for Facebook, Instagram,...* Obtenido de https://chatfuel.com/

Cho, K., van Merrienboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., & Schwenk, H. (2014). Learning phrase representations using RNN encoder–decoder for statistical machine translation. *arXiv:1406.1078.*

Chowdhury, G. G. (2003). Natural language processing. *Annual Review of Information Science and Technology, 37(1)*, 51-89.

Colby, K. M., Weber, S., & Hilf, F. D. (1971). Artificial paranoia. *Artificial Intelligence*, 1-25.

Collobert, R., & Weston, J. (2008). A unified architecture for natural language processing: deep neural networks with multitask learning. *Proceedings of the 25th International Conference on Machine Learning*, 160–7.

Despa, M. L. (2014). Comparative study on software development methodologies. *Database Systems Journal vol. V, no. 3*, 37-55.

F. X. (2019). *Premier AI Online Chatbot Software*. Obtenido de https://flowxo.com/

G. C. (2021). *About ChatterBot*. Obtenido de ChatterBot 1.0.8 documentation: https://chatterbot.readthedocs.io/en/stable/#

Google. (2019). *Google assistant, your own personal google.* Obtenido de https://assistant.google.com/

Google. (2021). *Dialogflow Documentation*. Obtenido de Google Cloud: https://cloud.google.com/dialogflow/docs/

Google. (14 de 12 de 2021). *Entidades | Dialogflow ES | Google Cloud.* Obtenido de Google Cloud: https://cloud.google.com/dialogflow/es/docs/entities-overview

Google. (30 de 11 de 2021). *Fulfillment | Dialogflow ES | Google Cloud.* Obtenido de Google Cloud: https://cloud.google.com/dialogflow/es/docs/fulfillment-overview

Hien, H. T., Cuong, P. N., Nam, L. N., Nhung, H. L., & Thang, L. D. (2018). Intelligent assistants in higher-education environments: The FIT-EBot, a chatbot for administrative and

learning support. *Proceedings of the ninth international symposium on information and communication technology*, 69–76.

IBM. (2020). *Intelligent Virtual Agent - IBM Watson Assistant*. Obtenido de https://www.ibm.com/cloud/watson-assistant/

IBM. (2021). *IBM Watson*. Obtenido de https://www.ibm.com/watson

International Diabetes Federation. (2019). *Atlas de la Diabetes de la FID, Novena Edición.*

Karami, A., Dahl, A. A., Turner-McGrievy, G., Kharrazi, H., & Shaw, J. G. (2017). Characterizing Diabetes, Diet, Exercise, and Obesity Comments on Twitter. *arXiv:1709.07916*.

Khanna, A., Pandey, B., Vashishta, K., Kalia, K., Bhale, P., & Das, T. (2015). A study of today's A.I. through chatbots and rediscovery of machine intelligence. *International Journal of U- and e-Service, Science and Technology, 8*, 277–284.

Kim, J., Lee, H.-G., Kim, H., Lee, Y., & Kim, Y.-G. (2018). Two-step training and mixed encoding-decoding for implementing a generative chatbot with a small dialogue corpus. 31–35.

Kompella, R. (9 de Febrero de 2018). *Conversational AI chat-bot - Architecture overview*. Obtenido de https://towardsdatascience.com/architecture-overview-of-a-conversational-ai-chat-bot-4ef3dfefd52e

Kucherbaev, P., Bozzon, A., & Houben, G. J. (2018). Human-aided bots. *IEEE Internet Computing, 22(6)*, 36-43.

Looije, R., Neerincx, M. A., & Cnossenc, F. (2010). Persuasive robotic assistant for health self-management of older adults: Design and evaluation of social behaviors. *Int. J. Human-Computer Studies 68*, 386–397.

Magyar, G., Balsa, J., Cláudio, A., Carmo, M., Neves, P., Alves, P., . . . Guerreiro, M. (2019). Anthropomorphic Virtual Assistant to Support Self-care of Type 2 Diabetes in Older People: A Perspective on the Role of Artificial Intelligence. *Proceedings of the 14th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications*, 323-331.

ManyChat, I. (2021). *Chat Marketing Made Easy With Manychat*. Obtenido de https://manychat.com/

Marietto, M., Varago de Aguiar, R., Barbosa, G., Botelho, W., Pimentel, E., Franca, R., & Silva, V. (2013). Artificial intelligence markup language: A brief tutorial. *International Journal of Computer Science and Engineering Survey, 04*.

McShane, M. (2017). Natural language understanding (NL, unot NLP) in cognitive systems. *AI Magazine, 38(4)*, 43-56.

Microsoft. (2019). *Cortana - Your personal productivity assistant.* Obtenido de https://www.microsoft.com/en-us/cortana

Microsoft. (2019). *Microsoft Bot Framework.* Obtenido de https://dev.botframework.com/

Microsoft. (2021). *LUIS (Language Understanding) - Cognitive Services - Microsoft*. Obtenido de https://www.luis.ai/home

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space.

Molnár, G., & Zoltán, S. (2018). The role of chatbots in formal education. *16th International Symposium on Intelligent Systems and Informatics (SISY)*.

Nayyar, D. A. (2019). *Chatbots and the open source tools you can use to develop them*. Obtenido de https://www.opensourceforu.com/2019/01/chatbots-and-the-open-source-tools-you-can-use-to-develop-them/

ngrok. (2022). Obtenido de ngrok - secure introspectable tunnels to localhost: https://ngrok.com/

Pallets. (2010). *Welcome to Flask*. Obtenido de Flask Documentation Website: https://flask.palletsprojects.com/en/2.0.x/

Pandorabots, I. (2008-2021). *Pandorabots: Home*. Obtenido de https://home.pandorabots.com/home.html

Petherbridge, N. (2009). *Artificial intelligence scripting language - RiveScript.com*. Obtenido de https://www.rivescript.com/

Postman. (2022). Obtenido de Postman API Platform: https://www.postman.com/

R. T. (2021). *Open source conversational AI*. Obtenido de RASA: https://rasa.com/

Radford, A., Wu, J., Child, R., Luan, D., & Amodei, D. (2019). Language Models are Unsupervised Multitask Learners.

Ramesh, K., Ravishankaran, S., Joshi, A., & Chandrasekaran, K. (2017). A survey of design techniques for conversational agents. *Information, communication and computing technology*, Vol. 750 (pp. 336–350).

S. C. (2021). *SAP Conversational AI*. Obtenido de Low Code Chatbot Building Platform: https://cai.tools.sap/

Singh, S., Darbari, H., Bhattacharjee, K., & Verma, S. (2016). Open source NLG systems: A survey with a vision to design a true NLG system. *9*, 4409–4421.

Sinoo, C., Pal, S. v., Henkemans, O. A., Keizer, A., Bierman, B. P., Looije, R., & Neerincx, M. A. (2018). Friendship with a robot: Children's perception of similarity between a robot's physical and virtual embodiment that supports diabetes self-management. *Patient Education and Counseling*, 1248–1255.

Spänig, S., Emberger-Klein, A., Sowa, J.-P., Canbay, A., Menrad, K., & Heider, D. (2019). The virtual doctor: An interactive clinical-decision-support system based on deep learning for non-invasive prediction of diabetes. *Artificial Intelligence In Medicine*.

Turchin, A., & Florez Builes, L. F. (2021). Using Natural Language Processing to Measure and Improve Quality of Diabetes Care: A Systematic Review. *Journal of Diabetes Science and Technology, 15(3)*, 553–560.

Wallace, R. S. (2009). The anatomy of a.l.I.C.e. *Parsing the turing test: philosophical and methodological issues in the quest for the thinking computer*, 181–210.

Weizenbaum, J. (1966). ELIZA—A computer program for the study of natural language communication between man and machine. *ACM*, 36–45.

Wilcox, B., & Wilcox, S. (2014). Making it real: Loebner-winning chatbot design. *Arbor, 189*, a086.

Wit.ai, I. (2020). *Wit.ai*. Obtenido de https://wit.ai/

Woolf, M. (18 de Octubre de 2021). *minimaxir/gpt-2-simple: Python package to easily retrain OpenAI's GPT-2 text-generating model on new texts*. Obtenido de GitHub: https://github.com/minimaxir/gpt-2-simple

Yale, J.-F., Paty, B., & A., P. (2018). Hypoglycemia. En D. C. Committee, *Diabetes Canada 2018 Clinical Practice Guidelines for the Prevention and Management of Diabetes in Canada.* (págs. 42(Suppl 1):S88-S103).

Zeng, G., Yang, W., Ju, Z., Yang, Y., Wang, S., Zhang, R., . . . Xie, P. (2020). MedDialog: Large-scale Medical Dialogue Datasets. *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*, 9241–9250.

Zhou, M., Duan, N., Liu, S., & Shum, H.-Y. (2020). Progress in Neural NLP: Modeling, Learning, and Reasoning. *Engineering*(6), 275-290.

Anexos

Anexo I. Repositorio de código del proyecto de Node.js

https://github.com/CPaulesS/MasterProject

Anexo II. Base de datos de conversaciones de Diabetes Mellitus creada manualmente

https://www.kaggle.com/carlospaules/diabetes-mellitus-daily-conversations

Anexo III. Repositorio de código del modelo en Python

https://github.com/CPaulesS/NLGmodel_DM

Anexo IV. Notebook de Google Colab del entrenamiento del modelo de 124 millones de parámetros

https://colab.research.google.com/drive/1QVoG4FYV1ZdGVhppohUi77IMcYmnKErY?usp=sharing

Anexo V. Artículo de investigación

Agente conversacional para registrar información de pacientes con diabetes mellitus tipo 1

Carlos Paules Sánchez

Universidad Internacional de la Rioja, Logroño (España)

Fecha: 2 de febrero de 2022

Resumen

En este proyecto se ha creado un agente conversacional usable y funcional empleando Dialogflow, que utiliza técnicas de procesamiento del lenguaje natural para mantener conversaciones diarias con usuarios diabéticos tipo 1. La información sobre diabetes detectada se almacena en la base de datos Firebase. Se ha seguido una metodología iterativa e incremental para el diseño, implementación y testeo del agente con cuatro usuarios. El agente almacena información sobre niveles de glucosa, dosis de insulina, ejercicio físico, comida y situaciones causantes de estrés. También se ha creado una metodología para implementar respuestas dinámicas que finalmente no se ha podido aplicar por limitaciones de Dialogflow. La puntuación System Usability Scale del agente ha sido 85,5 demostrando que es una herramienta usable. La mitad de respuestas generadas por el modelo son adecuadas en la conversación excepto en mensajes de glucosa e insulina debido al reducido tamaño de la base de datos de entrenamiento.

Palabras Clave

agente conversacional, diabetes mellitus tipo 1, procesamiento del lenguaje natural, creación de bases de datos, Dialogflow

I. Introducción

La Diabetes Mellitus (DM) es una enfermedad crónica que en la actualidad afecta a un 9,3% de la población mundial adulta [1] y cuya prevalencia ha ido aumentando anualmente en adultos y niños. Existen varios tipos de DM de los cuales destacan dos: la DM tipo 1 y la DM tipo 2. Este proyecto se ha centrado en la DM tipo 1, que provoca una alteración del nivel de glucosa en sangre debida a un déficit de generación de insulina. El buen control diario del nivel de glucosa en sangre es esencial para evitar posibles complicaciones a corto y largo plazo.

El interés por la investigación y desarrollo de sistemas que utilizan inteligencia artificial ha ido en aumento en los últimos años, haciendo que actualmente herramientas software como los agentes conversacionales sean muy populares. Estos agentes se pueden diseñar para mantener conversaciones de temáticas diversas o llevar a cabo funciones de distinto tipo. En este proyecto se ha pensado que un agente conversacional puede aportar mucho valor en la gestión de la DM manteniendo conversaciones diarias con un usuario en las que se puedan recoger y aportar datos sobre el control de la enfermedad.

Por ello, se ha decidido crear un agente conversacional usable y funcional que emplee técnicas de procesamiento del lenguaje natural (PLN) para captar y almacenar información sobre la DM en conversaciones de chat diarias con usuarios diabéticos. Para ello, ha sido necesario realizar un estudio a fondo sobre la DM, hallando los factores que más influyen diariamente en el control de la glucosa en sangre. También ha sido necesario estudiar todo el proceso de creación de agentes conversacionales, los distintos tipos que existen, las herramientas que existen actualmente para desarrollarlos, las técnicas de PLN que utilizan y su funcionamiento. Una vez terminado el estudio del contexto se han extraído unas conclusiones parciales en las que se han determinado las herramientas más adecuadas para desarrollar el proyecto: Telegram para la interfaz gráfica, Dialogflow para el desarrollo y Firebase como base de datos. Posteriormente se han establecido los objetivos, los requisitos que se deben cumplir y la metodología más adecuada para la creación de un agente conversacional de este tipo. Llegados a este punto se ha comenzado con la implementación del agente conversacional, conectando las distintas plataformas que el agente emplea y creando una primera versión capaz de almacenar información sobre DM en una base de datos y de generar respuestas predefinidas. Seguidamente se ha procedido a implementar las respuestas dinámicas creadas por un modelo de generación de lenguaje natural (GLN), el cual es llamado por el agente durante una conversación con un usuario. Esta metodología no ha dado los resultados esperados debido a una limitación del tiempo de respuesta de Dialogflow. Finalmente se ha evaluado la usabilidad del agente conversacional con respuestas predefinidas usando la *System Usability Scale* y se ha obtenido una puntuación media de 85,5, demostrando que la herramienta creada es usable. También se ha evaluado el funcionamiento del modelo y se ha observado que la mitad de respuestas generadas a mensajes de eventos que influyen en el nivel de glucosa son correctas, mientras que sólo un 30-35% de las respuestas generadas a mensajes sobre niveles de glucosa y dosis de insulina son correctas. Esto ha mostrado que es necesario entrenar el modelo con más conversaciones, especialmente algunas con información de DM.

II. Estado del Arte

La DM es una enfermedad crónica que genera un aumento de concentración de glucosa en sangre debido a un déficit de insulina o a un mal uso de ésta. La insulina es una hormona encargada de retirar la glucosa del torrente sanguíneo para introducirla en células que requieran energía, donde la glucosa se metaboliza. Un mal control de esta enfermedad puede generar complicaciones a largo plazo como enfermedades cardiovasculares, problemas renales, degeneración del sistema nervioso y afectaciones oculares [1]. La enfermedad de afectación del sistema nervioso

más conocida es el "pie diabético", que consiste en una pérdida de sensibilidad del pie y de la capacidad curativa de la sangre, provocando úlceras. Tal y como se ha comentado en la introducción, este proyecto se centra en la DM tipo 1, que es debida a una reacción autoinmune del cuerpo a las células que producen insulina haciendo que éstas reduzcan la producción hasta detenerse por completo. Las causas de esta reacción son desconocidas, pese a que se sospecha que está influenciada por una predisposición genética. Existen varios tratamientos para la DM tipo 1, pero el más usado es la inyección de insulina combinado con una dieta controlada y un estilo de vida saludable [2]. Además de las complicaciones a largo plazo, la DM tipo 1 tiene dos riesgos a corto plazo causados por un mal control de la glucosa en sangre: la hiperglucemia y la hipoglucemia. La hiperglucemia es un estado de glucosa en sangre elevado que causa deshidratación y micción frecuente. La hipoglucemia es un estado de concentración de glucosa en sangre baja que suele causar mareos y en casos graves puede ser muy peligrosa. La hiperglucemia se suele tratar haciendo ejercicio físico o inyectando una dosis de insulina de acción rápida, mientras que la hipoglucemia depende del grado. Para tratar una hipoglucemia leve se recomienda la ingestión de 15g de carbohidratos, que pueden ser azúcar o bebidas azucaradas. Para una hipoglucemia grave se requiere la inyección de glucagón, la hormona antagonista de la insulina que libera glucosa al torrente sanguíneo [3].

El PLN es una rama de la inteligencia artificial centrada en el reconocimiento y comprensión del lenguaje humano por parte de sistemas computacionales. Esta disciplina combina la ciencia computacional con la lingüística logrando que computadoras puedan comprender los significados (semántica) y las reglas (sintaxis) del lenguaje. El PLN tiene distintas aplicaciones de las cuales destacan la categorización de contenido de un texto, el análisis de sentimiento, la transformación de lenguaje oral a escrito y viceversa, la detección de duplicaciones, la generación de textos, la traducción automática y el resumen de textos. La técnica principal detrás del PLN se denomina *embedding*, y consiste en mapear palabras u oraciones a un espacio semántico convirtiéndolas en vectores, que estarán más cerca cuanto mayor sea la similitud de su significado. Estos vectores se pueden introducir posteriormente en un modelo de aprendizaje profundo que devuelve un resultado distinto dependiendo del objetivo del PLN [4]. Las técnicas para el cálculo de *embeddings* son distintas en función de si se considera el contexto de la palabra. Cuando se tiene en cuenta el contexto se utilizan redes neuronales para el cálculo, mientras que cuando no se considera se utilizan técnicas más simples que no están alineadas con el objetivo de este proyecto. Para los *chatbots* se suele utilizar el modelo codificador-decodificador con secuencias enteras [5]. Este tiene dos redes neuronales recurrentes que actúan como codificador y decodificador respectivamente. La primera analiza cada palabra de la oración de entrada y va creando un estado oculto con información de la oración analizada hasta el momento. La segunda recoge la información creada por el codificador de la última palabra y va creando palabras hasta llegar a un elemento final.

Una búsqueda de estudios focalizados en el PLN y la DM ha mostrado principalmente dos tipos distintos en función del objetivo. Unos están centrados en crear herramientas que emplean técnicas de PLN con distintos motivos: diagnosticar la DM, detectar hipoglucemias o estudiar la influencia del estilo de vida en la enfermedad. Los otros tienen como objetivo realizar una investigación clínica utilizando técnicas de PLN [6]. El objetivo de este proyecto está alineado con el primer tipo de estudios.

Una de las aplicaciones del PLN más usadas en la actualidad son los agentes conversacionales, denominados en inglés *chatbots*. Un *chatbot* es un programa que utiliza técnicas de PLN para mantener conversaciones de temáticas diversas o más específicas con humanos u otros *chatbots*. Para el desarrollo de agentes conversacionales se utilizan principalmente dos técnicas distintas: *pattern matching* y aprendizaje automático. La primera es un algoritmo basado en reglas, en la cual el agente compara los mensajes recibidos con las reglas de su base de datos y devuelve la respuesta predefinida establecida. La segunda consiste en un modelo de aprendizaje automático que utiliza PLN y al recibir el mensaje del usuario busca clasificar su intención en alguno de los *intents* configurados en el agente. Esta tiene en cuenta el contexto y puede extraer información específica del mensaje, además de ser capaz de generar respuestas dinámicas. La estructura interna un *chatbot* se compone de cinco partes principalmente: la interfaz de usuario, el módulo de análisis de mensaje del usuario, el gestor de diálogo, el *backend* y el módulo de generación de respuesta [7]. La interfaz de usuario es el medio por donde se lleva a cabo la conversación, para lo cual se suelen utilizar aplicaciones de mensajería instantánea. El componente de análisis de mensaje contiene un módulo de comprensión de lenguaje natural que extrae la intención del mensaje y toda la información deseada [8]. El gestor de diálogo se ocupa de mantener la intención actualizada y la información del contexto, y por tanto decido cómo continuar la conversación en función de la intención detectada [9]. El componente de *backend* tiene funciones distintas dependiendo del objetivo del agente conversacional, desde contener la base de datos donde almacenar o extraer información hasta una API a la que se pueden hacer llamadas para hacer solicitudes, reservas o búsquedas, entre otras cosas [10]. El módulo de generación de respuesta puede utilizar un modelo basado en reglas con respuestas predefinidas o puede usar un modelo de GLN que emplee redes neuronales recurrentes para crear una respuesta automática [11]. Se han buscado bases de datos de conversaciones con información sobre la DM para entrenar un modelo de GLN en este proyecto y, puesto que no se ha encontrado ninguna de utilidad, se ha decidido crear una manualmente.

Existen generalmente dos tipos de plataformas para desarrollar agentes conversacionales, unas de fuente abierta que ofrecen control completo en la implementación, y otras plataformas comerciales que no permiten acceso total al código, pero ofrecen más facilidades a la hora de desarrollar el agente. Algunos ejemplos de plataformas de fuente abierta son Botkit [12], RASA [13], Chatterbot [14] o Microsoft Bot Framework [15], y ejemplos de plataformas comerciales son Dialogflow [16], LUIS [17], IBM Watson [18] o Amazon Lex [19].

Las distintas investigaciones con objetivos alineados con este proyecto que se han estudiado se pueden diferenciar dos tipos: la creación de bases de datos con información de DM y la creación de asistentes virtuales que puedan aportar valor en la gestión de la DM. En referencia a la creación de bases de datos se ha observado que la mayoría de investigaciones en las que se han creado modelos predictivos relacionados con la DM, el objetivo de los modelos era el diagnóstico de la DM o la detección de complicaciones [20]. Esto implica que las bases de datos creadas para los modelos de diagnóstico contienen información anatómica de personas y posibles factores que influyen en la DM, y las de los modelos de detección de complicaciones contienen imágenes de ojos con y sin retinopatías.

Los estudios centrados en la creación de asistentes virtuales para aportar valor a la DM se diferencian en dos, unos que consisten en aplicaciones que centralizan toda la información sobre la enfermedad y otros que se basan en la creación de agentes conversacionales para ayudar en la gestión de la DM. Este

proyecto está más alineado con el segundo tipo de asistentes, y por tanto se han estudiado únicamente los de este tipo.

Inicialmente se han hallado dos estudios de un grupo de investigación que estaban centrados en encontrar qué interfaz tecnológica era más accesible y confiable para los usuarios. En el primer estudio [21] se utilizaban tres interfaces distintas para mantener conversaciones sobre diabetes con usuarios: la primera era una interfaz de texto, la segunda era un avatar virtual y la tercera era un avatar físico. Las conversaciones que estos asistentes mantenían estaban controladas por los investigadores, y por tanto no eran autónomos. Se concluyó que las interfaces con mayor aceptación fueron las de texto escrito y el avatar virtual. El segundo estudio [22] era análogo al primero, pero se utilizó únicamente un avatar virtual y otro físico, y se evaluó con niños. En este caso se mostró que la combinación entre el avatar virtual y físico era la mejor solución.

Otro grupo de investigación creó un asistente robótico con una infraestructura para registrar información y ayudar en la gestión de la DM de niños [23]. Este robot estaba conectado a un sistema en internet al cual tenían acceso los adultos responsables del niño y los especialistas médicos. El asistente mantenía conversaciones adaptadas a cada paciente en las cuales se sugería una gestión óptima de la medicación basada en la información de glucosa recopilada por sensores (que se enviaba directamente al sistema) y otro tipo de información captada durante la conversación. El foco de este proyecto era la coordinación entre el paciente, la persona responsable y el especialista médico para una mejor gestión de la DM.

En otro estudio se creó un asistente virtual antropomórfico para dar soporte a enfermos con DM tipo 2 mayores de 65 años [24]. El objetivo de este asistente era generar un cambio del estilo de vida del usuario que contribuyese en una mejor gestión de la enfermedad. Esto se hacía mediante conversaciones de texto y voz con el usuario en que el agente promovía la auto medicación, la actividad física y un control de la dieta. Existía la posibilidad de introducir información manualmente en la herramienta, que se almacenaba junto con la información extraída de la conversación en una base de datos y se utilizaba para adecuar los diálogos del agente.

Se ha encontrado también un estudio en el cual se creó asistente virtual cuyo objetivo principal era la detección y diagnóstico de la enfermedad [25]. Este consistía en una cabina con aparatos para registrar datos del paciente que además contenía un módulo conversacional de voz. La información registrada por los aparatos y recibida en la conversación se introducía en modelos de aprendizaje profundo y máquinas de vector soporte que realizaban un análisis probabilístico de los datos y devolvían un diagnóstico. Este diagnóstico mostraba la probabilidad de padecer la DM además de un informe con la información extraída.

De todo este estudio del contexto y estado del arte se ha concluido que el foco de los asistentes es el diagnóstico o una mejor gestión de la DM, pero ninguno se centra en el almacenamiento de datos. Por tanto, para llevar a cabo este proyecto se ha decidido utilizar Dialogflow para el desarrollo del agente, Telegram como interfaz gráfica y Firebase como base de datos. Telegram se ha escogido por ser una aplicación de mensajería de las más utilizadas en la actualidad y que tiene un método de integración con agentes conversacionales muy sencillo. Se ha elegido Dialogflow para el desarrollo porque ofrece muchas posibilidades para la implementación del agente y de forma mucho más sencilla que los *frameworks* de fuente abierta. También se ha escogido Firebase porque es la base de datos pensada para Dialogflow y también desarrollada por Google, cosa que facilita enormemente la conexión de ambas aplicaciones. Estas herramientas contienen métodos de creación de *chatbots* que utilizan aprendizaje automático y *embeddings*. Finalmente, se ha decidido crear una metodología de llamadas a un modelo de GLN para crear respuestas dinámicas a mensajes con información de DM. Este modelo se entrenará inicialmente con una base de datos de conversaciones con información de DM creada manualmente, y posteriormente se utilizarán los registros de conversaciones del agente para el reentrenamiento.

III. Objetivos y Metodología

El objetivo principal de este proyecto es la creación de un agente conversacional que mantenga conversaciones diarias con usuarios diabéticos tipo 1 y almacene la información relacionada con la enfermedad en una base de datos. Para ello, el agente debe utilizar técnicas de PLN para ser capaz de comprender los mensajes del usuario, extraer la información adecuada, almacenarla en la base de datos y generar una respuesta adecuada. El agente se ha de evaluar en términos de su usabilidad y su funcionalidad.

Para poder llevar este proyecto a cabo y cumplir la meta establecida, ésta se debe dividir en cuatro objetivos específicos. El primero, realizado en el apartado anterior, consiste en el estudio del arte de agentes conversacionales centrados en DM y en todas las tecnologías que se utilizan para su desarrollo. Además, se debe identificar la información más relevante para la gestión de la DM para poder almacenarla. El segundo objetivo es diseñar el agente conversacional utilizando toda la información acerca de las técnicas y herramientas estudiada anteriormente. El siguiente objetivo es implementar el agente siguiendo el diseño creado y haciendo las adaptaciones necesarias en caso de que aparezcan dificultades. Finalmente, se pretende evaluar la herramienta en términos de su funcionalidad y su usabilidad.

La metodología elegida para crear el agente y cumplir los objetivos definidos previamente es la iterativa e incremental. La base de esta metodología es la creación de un modelo inicial con un conjunto de especificaciones que se va evaluando y expandiendo hasta llegar a crear un producto definitivo. Para ello se debe seguir un número de pasos o etapas, algunas de las cuales se deben realizar de forma iterativa.

La primera etapa que se ha realizado es una investigación de los requerimientos básicos y de los objetivos que la herramienta software debe cumplir. Para poder llevar a cabo esta etapa correctamente se ha estudiado previamente toda la documentación necesaria para desarrollar un agente conversacional y las investigaciones en este campo de índole similar. Seguidamente se ha pasado a la primera iteración de un conjunto de etapas que se repetirán hasta llegar a crear el producto final.

La primera etapa del proceso iterativo es la planificación, en la cual se ha definido el flujo que debe seguir la aplicación y se ha dividido en procesos distintos con diferentes funcionalidades. El desarrollo del agente conversacional se ha dividido en tres partes que utilizan herramientas software distintas: la configuración de la interfaz gráfica, la configuración del agente conversacional y la configuración de la base de datos. La conexión y el correcto funcionamiento de estas partes ha permitido crear un agente conversacional capaz de tener conversaciones diarias con usuarios. La siguiente etapa es el diseño de la apariencia gráfica de la aplicación, un proceso trivial en este proyecto debido a que se ha utilizado Telegram como interfaz gráfica para el agente. Posteriormente se ha procedido al desarrollo del agente conversacional utilizando Dialogflow y configurando todos los

módulos de conversación del agente. En esta etapa también se ha configurado las entregas asociadas a los mensajes de las conversaciones, mediante las cuales es posible almacenar y retirar información de la base de datos o implementar las respuestas dinámicas creadas por un modelo de GLN. Una vez implementado el agente se ha evaluado para comprobar que cumple con los objetivos establecidos, y en caso contrario identificar los problemas y solucionarlos. Esta evaluación ha proporcionado un *feedback* que ha mostrado que funcionalidades se deben añadir y cómo se debe mejorar la herramienta para que se considere un producto finalizado. En base a este *feedback*, se debe volver al estado de planificación comenzando la segunda iteración del proceso.

Una vez realizadas las iteraciones necesarias para obtener un producto final, se realizaría la instalación en un ambiente real. Esto no es necesario en este proyecto debido a que ya se encuentra en ese estado desde el principio. Por tanto, se procederá a la última etapa que es el mantenimiento, en la cual se comprobará que el agente conversacional funcione siguiendo el comportamiento esperado y se actualizará o modificará si es necesario.

IV. Contribución

La creación de un nuevo agente conversacional ha comenzado con un registro en la web de Dialogflow utilizando una cuenta de Google. Tras este registro se ha podido crear un nuevo agente, para el cual ha habido que configurar su nombre, su idioma principal y los secundarios, además de asignar un proyecto de Google Cloud Functions. En caso de no tener un proyecto de este tipo se crea automáticamente uno con el mismo nombre que el agente. Este proyecto conecta todas las aplicaciones creadas en *frameworks* de Google como Dialogflow o Firebase, y permite ver un análisis del uso de las aplicaciones o configurar un plan de facturación entre otras opciones. El idioma principal elegido para el nuevo agente creado ha sido el inglés debido a que es más sencillo encontrar bases de datos para entrenar modelos de generación de texto en ese idioma, pero también se ha configurado en español. La implementación del agente en este *framework* se basa en la configuración de cuatro elementos clave: los *intents*, las entidades, las entregas y las integraciones.

Un *intent* corresponde a la intención o información que se quiere transmitir con un mensaje. El agente utiliza estos *intents* como clases en las cuales se puede asignar un mensaje de un usuario. Al activarse un *intent*, el agente realizará las funciones configuradas en ese *intent*, ya sea la recopilación de información, el almacenamiento de la misma en una base de datos o la generación de una respuesta. La información recopilada en un mensaje se guarda en un parámetro, que tiene un tipo de entidad que indica cómo se obtienen estos datos del mensaje. Existen entidades definidas por el sistema como son números, fechas, colores, etc., o también se pueden crear entidades personalizadas para extraer otro tipo de información específica como por ejemplo un estado de glucosa en sangre o una actividad física realizada. Activar las entregas para un *intent* hace que cuando éste se active, el agente pueda realizar funciones definidas en un archivo Node.js tales como el almacenamiento de información en una base de datos, la extracción o búsqueda de información, o generar una respuesta dinámica en lugar de la predefinida en Dialogflow. Finalmente, la integración se basa en el despliegue del agente a plataformas de conversación telefónica o escrita.

Tras la creación del agente en Dialogflow, se ha procedido a crear un nuevo *bot* en Telegram, aplicación que sirve de interfaz gráfica para el agente. Esto se ha llevado a cabo iniciando una conversación con BotFather, un *bot* de Telegram centrado en la gestión de *bots* en esta plataforma. En esta conversación se ha configurado el nuevo *bot* y se ha obtenido un token único que, tras introducirlo en el apartado de integraciones, permite la conexión con el agente de Dialogflow.

La tercera aplicación que se ha usado en este proyecto y que se ha tenido que conectar al agente es la base de datos. Para este proyecto se ha utilizado Firebase, una aplicación para el almacenamiento de información desarrollada también por Google. Esto ha facilitado en gran medida el uso de esta herramienta, su conexión con el agente conversacional y la gestión de la base de datos. De los dos tipos de bases de datos disponibles dentro de Firebase se ha elegido Firestore Database debido a que es la versión más nueva, tiene un modelo de datos más intuitivo y permite realizar consultas más rápidas. Para realizar la conexión únicamente ha sido necesario crear la base de datos usando el mismo proyecto de Google Cloud Functions que utiliza el agente de Dialogflow y escoger el plan de facturación gratuito.

Una vez creada la base de datos, se ha activado las entregas para el agente en Dialogflow y se ha habilitado el servicio de *webhook* que automáticamente se completa con el enlace a Firebase. Para trabajar con el servicio de *webhook* en las entregas ha sido necesario descargar la API de Firebase a la máquina local, instalarla e inicializarla especificando la configuración y las funcionalidades necesarias para un correcto funcionamiento de la aplicación. También se han descargado el archivo donde se configuran las entregas y que contiene todo el código y funciones, index.js, y el archivo que contiene las dependencias, package.json. En el archivo index.js es donde se inicializa la base de datos y se realiza el mapeo de los *intents* del agente a sus funciones correspondientes.

Teniendo todo el sistema conectado, se ha comenzado con la creación y configuración de *intents* para las conversaciones. Los únicos *intents* que aparecen en un agente nuevo son el de saludo y el de *fallback*. El primero se corresponde al inicio de una conversación con el agente, y el segundo se activa cuando el agente no es capaz de asignar un mensaje de usuario a ningún otro *intent*. El primer paso ha sido importar Smalltalk, un agente prediseñado por Google con *intents* para conversaciones casuales que hacen al agente un poco más personal. Este agente importado contiene un conjunto de *intents* para los cuales se pueden configurar respuestas personalizadas o dejar las que vienen por defecto. Posteriormente se ha activado las entregas para el *intent* de saludo de tal forma que si el usuario nunca ha hablado con el agente se inicia una conversación de introducción, mientras que si ya ha habido contacto previamente el agente hace un saludo cercano y una pregunta casual. Esto es posible gracias a la comprobación de la existencia de datos del usuario en la base de datos.

La conversación de introducción para usuarios nuevos está formada por cinco preguntas que activan cinco *intents* de forma consecutiva. En esta conversación se obtiene información sobre el nombre del usuario, su edad, la edad a en la cual le diagnosticaron la diabetes, el tipo de diabetes y el tipo de tratamiento que sigue. Todos estos *intents* tienen las entregas activadas y tienen funciones que extraen la información necesaria del mensaje, la almacenan en la base de datos y generan la siguiente respuesta configurada. El almacenamiento se realiza mediante una función que hace una llamada a la base de datos, busca la colección correspondiente al ID de Telegram del usuario y el documento de información básica y escribe la información en un formato [clave]: valor. En caso de que la colección o el documento no existan, Firestore los crea automáticamente.

Después de una primera conversación para adquirir la información básica, se han creado y configurado un número de *intents* que aportan información directa sobre la gestión de la DM

u otro tipo de información que puede afectar a los niveles de glucosa del usuario. La información recogida y almacenada en estos *intents* es sobre niveles de glucosa, dosis de insulina, comida ingerida, actividad física realizada y situaciones que generan estrés al usuario. Toda la información de este tipo detectada por el agente en una conversación se almacena en un documento de Firestore dentro de la colección del usuario. El nombre del documento es la fecha del día de la conversación en formato DD-MM-AAAA, y por tanto se crea un documento nuevo cada día que hay una conversación en la cual se activan estos *intents*. Esta fecha se extrae del mensaje recibido en Telegram en formato de instante de tiempo Unix, y se transforma al formato de fecha utilizando una función que hace esta conversión.

La configuración de cada uno de los *intents* de recopilación de información sobre la DM es distinta, pero las funciones asociadas a cada uno en el index.js tienen una estructura similar. Para detectar información sobre el nivel de glucosa en sangre se ha creado una entidad que corresponde al estado de glucosa en sangre (alta, baja o buena). Con esta entidad y otras del sistema se recoge información sobre el nivel numérico o estado de glucosa en sangre y el instante de tiempo en el cual se ha detectado. Para el *intent* que recoge información sobre la dosis de insulina administrada también se ha tenido que crear una nueva entidad que corresponde al tipo de insulina inyectada, y el resto se han utilizado las del sistema. Así, en estos mensajes se recoge información sobre la dosis de insulina inyectada, el tipo de insulina y el instante de tiempo en que se ha inyectado. Los mensajes con información sobre una actividad física realizada se clasifican en dos *intents* distintos para añadir más variabilidad en la conversación con el usuario. Para captar la información en estos mensajes se ha creado una entidad nueva que contiene una larga lista de tipos de deporte distintos. Gracias a esto se puede recoger información en el mensaje sobre el tipo de ejercicio físico, la duración del mismo y el instante de tiempo en que se ha practicado. El *intent* que recoge información acerca de la comida ingerida también ha requerido definir en este caso dos entidades nuevas para tipos de comida con alto y con bajo nivel de carbohidratos. Esto se ha realizado para almacenar información más precisa de este tipo de eventos en la base de datos. Esta información recopilada consiste en el tipo de comida, el instante de tiempo en que se ha comido y la cantidad que se ha ingerido en peso o número. Para el último *intent* creado no ha sido necesario definir ninguna entidad nueva, debido a que únicamente se recopila información acerca de un instante de tiempo en el que sucede un evento que causa estrés al usuario.

Todas las funciones asociadas a estos *intents* siguen la misma estructura. Inicialmente se comprueba qué información hay para almacenar en la base de datos y se guarda en variables, ya sea la fecha o los distintos tipos de información descritos anteriormente. Después se actualiza un contador de eventos diarios y finalmente se llama a la función que almacena la información sobre la DM en Firestore. Esta información se almacena en eventos, y por tanto para cada día puede haber un número de eventos distinto de cada tipo, y dentro de cada evento se encuentra toda la información recopilada en la conversación. Finalmente se han creado respuestas predefinidas para cada *intent* con consejos para la gestión de la enfermedad que hacen avanzar la conversación, aunque la intención es implementar respuestas dinámicas generadas por un modelo. Un ejemplo de la conversación de un usuario con el agente usando estas respuestas predefinidas se puede observar en la Figura 1.

Para poder implementar las respuestas dinámicas en el agente es necesario un modelo de GLN capaz de crear una respuesta a partir de un mensaje de entrada. Debido a que la creación y entrenamiento de un modelo nuevo es un proceso para el cual se requiere una gran cantidad de datos y una capacidad de cómputo

Fig. 1. Ejemplo de una conversación diaria del agente con un usuario diabético.

muy elevada, se ha decidido utilizar GPT-2 para este proyecto [26]. GPT-2 es un modelo de PLN basado en transformadores capaz de generar muestras de texto de alta calidad, que fue desarrollado por OpenAI. Este modelo funciona prediciendo cada siguiente palabra a partir de todas las palabras previas del texto.

Este modelo se ha importado y utilizado en este proyecto utilizando un paquete de Python que contiene funciones que simplifican su manejo. Este paquete se denomina gpt-2-simple [27] y está configurado para utilizar la GPU al trabajar con el modelo, haciendo que sea un proceso más rápido y eficiente. Se han utilizado principalmente cuatro funciones de este paquete. La primera para descargar los modelos deseados de entre cuatro disponibles (124, 355, 774 y 1558 millones de parámetros). La segunda para reentrenar el modelo, la tercera para cargar un *checkpoint* de un entrenamiento pasado y la cuarta para generar texto, a la cual se le puede introducir un prefijo.

Para que este modelo fuera capaz de generar texto en formato de diálogos ha sido necesario entrenarlo con numerosos datos de este mismo formato. Estos datos se han obtenido de una base de datos de conversaciones de actualidad pensada para entrenar *chatbots* extraída de Kaggle: *Chatbot Dataset Topical Chat* [28]. Para que el modelo fuera capaz de generar respuestas a mensajes

con información de DM, se ha creado una pequeña base de datos con simulaciones de conversaciones cuotidianas entre usuarios en las que aparece información sobre la DM. Inicialmente se han escogido los modelos de 124 y 355 millones de parámetros para el entrenamiento debido a que son los que menos tiempo tardan en generar la respuesta. Ambos se han entrenado primero durante 1000 pasos con la base de datos extraída de Kaggle para aprender a generar texto en formato de conversación, y posteriormente se han entrenado durante 100 pasos con menor *learning rate* utilizando la base de datos creada.

Tras haber entrenado el modelo se han creado dos funciones adicionales para adecuar el formato de los mensajes de entrada y salida del modelo. En los mensajes de entrada se han añadido signos de puntuación en caso de que no los hubiera, y las respuestas generadas se han reducido a un único mensaje si el modelo generaba más turnos de conversación.

Para poder acceder al modelo desde un entorno distinto ha sido necesario crear una *script* de Python con el código de generación de respuestas del modelo. Esta *script* se ha convertido en una aplicación web y se ha asignado a un puerto local utilizando la aplicación Flask [29]. El acceso a la aplicación web desde Firebase es posible gracias a ngrok, una aplicación que hace público el *localhost* creando una URL dinámica que apunta al puerto local especificado previamente.

Finalmente se ha creado en el código de las entregas de Dialogflow una función para hacer llamadas al modelo que se ejecuta en la máquina local. Esta función convierte el mensaje del usuario a un formato JSON y utilizando una función del paquete *request-promise* y el método POST envía el mensaje a la URL dinámica que apunta al modelo y muestra por pantalla la respuesta recibida. A pesar de que toda esta configuración e implementación funciona correctamente, una limitación de Dialogflow impide que la respuesta generada se muestre por la aplicación de mensajería. Esto es debido a que Dialogflow tiene definido un *timeout* de 5 segundos para enviar una respuesta, y si no se recibe ninguna respuesta en el código de entregas en ese tiempo el agente envía la respuesta predefinida en la configuración del *intent*. El tiempo de generación de respuesta del modelo más pequeño es como mínimo de 6 a 8 segundos, que aumenta si se tiene en cuenta el tiempo de envío y recepción de información. A pesar de ello, se pueden observar las respuestas dinámicas generadas por el modelo en el registro de Firebase.

Se ha realizado una búsqueda en la web para superar esta limitación y poder crear un agente totalmente funcional, y se han hallado dos posibles soluciones. La primera es desarrollar el agente en Dialogflow CX, una versión distinta del *framework* de creación de agentes conversacionales que permite aumentar el *timeout* a 30 segundos. Esta opción se ha descartado porque Dialogflow CX trabaja con un modelo de datos distinto, lo cual dificulta en gran medida una migración del proyecto y requeriría la creación de un nuevo proyecto análogo en este *framework*. La segunda solución consiste en hacer llamadas a dos eventos de seguimiento en cadena mientras no se haya recibido una respuesta, aumentando el límite a 15 segundos. Esta solución tampoco se ha podido implementar debido a la configuración de las llamadas al modelo en el archivo index.js. Para que el funcionamiento sea correcto y la función asociada al *intent* espere la respuesta del modelo, la llamada al modelo debe hacerse como retorno de la función del *intent*, pero esto impide que se puedan llamar eventos de seguimiento. La única solución sería utilizando una base de datos intermedia, pero por limitaciones de tiempo y dificultad a la hora de crear esta infraestructura se ha desestimado.

V. Evaluación y Resultados

En los objetivos del proyecto se han definido las dos cualidades a valorar del agente conversacional: la funcionalidad y la usabilidad. Debido a la limitación de Dialogflow comentada previamente que ha impedido que los usuarios puedan observar las respuestas del modelo de GLN, se ha evaluado el agente conversacional con respuestas predefinidas y el modelo por separado. Se ha evaluado la usabilidad del agente utilizando la *System Usability Scale* (SUS) [30], una de las escalas más utilizadas para valorar el uso de una herramienta o sistema de software. La funcionalidad del modelo de GLN también se ha evaluado por separado usando un método de evaluación centrado en el ser humano, en la cual se ha juzgado la validez de las respuestas generadas por el modelo a mensajes reales de usuarios.

Evaluación 1: SUS del agente conversacional

La SUS se considera en la actualidad un estándar para la valoración de la usabilidad de sistemas computacionales. Consiste en un cuestionario de 10 preguntas simples cuyas respuestas se realizan en una escala de 5 opciones, desde totalmente en desacuerdo (1) hasta totalmente de acuerdo (5). En este cuestionario, las preguntas impares tienen un tono positivo y las pares un tono negativo. Para calcular la puntuación final se suman las puntuaciones de cada pregunta, que si es impar se calcula como 1 menos el valor de la respuesta y si es par como 5 menos el valor de la respuesta, y se multiplica por 2,5. Esto genera una puntuación final del 0 al 100 pero que no se corresponde con un porcentaje, ya que 80 o mayor se considera una muy buena usabilidad, 68 sería la media y menor de 50 implica serios problemas de usabilidad.

La herramienta software desarrollada se ha evaluado utilizando la SUS por cuatro usuarios que padecen DM tipo 1 desde hace más de 10 años. La primera usuaria ha valorado el sistema con una puntuación SUS de 75, un valor por encima de la media. Las cuestiones con una puntuación más baja de esta usuaria han sido una que indica que hay demasiada inconsistencia en el sistema y otra que hace referencia a la seguridad del usuario usando el sistema. La valoración personal de esta usuaria ha sido que la idea del proyecto es buena y que si la herramienta llegase a un estado más evolucionado podría ayudar a muchos diabéticos.

El cuestionario del segundo usuario ha sumado una puntuación SUS de 80, lo cual indica un buen nivel de usabilidad. En este caso las valoraciones más bajas han sido para la frecuencia con la que se querría usar el agente conversacional y la seguridad del usuario al utilizarlo. El usuario ha valorado personalmente el agente conversacional diciendo que es una buena herramienta para hablar de diabetes, pero a medida que se usa las conversaciones tienden a volverse repetitivas.

La valoración SUS de la tercera usuaria a la herramienta software ha sido de 95, la más alta. La menor puntuación de esta usuaria ha sido en las cuestiones sobre frecuencia de uso y seguridad al usarlo, al igual que el usuario anterior. Su valoración personal del agente ha sido muy positiva, remarcando el gran potencial de la aplicación y la utilidad de los recordatorios que ayudan en la gestión de la DM.

La última usuaria ha valorado la usabilidad del agente conversacional con una puntuación SUS de 92,5, muy positiva. Las cuestiones que han recibido una menor puntuación por parte de esta usuaria han sido las mismas que las de la primera usuaria, la incoherencia en el sistema y la seguridad que siente la usuaria al usarlo.

Tras analizar todas las puntuaciones SUS se obtiene que la valoración media de la usabilidad del sistema es 85,5, cosa que demuestra que la herramienta creada es usable. Si se realiza un análisis global de las cuestiones con menor puntuación se observa que la valoración más baja ha sido la seguridad que siente el usuario usando el sistema, seguido de la incoherencia del agente

TABLA I

CLASIFICACIÓN DE LAS RESPUESTAS GENERADAS POR EL MODELO A LOS 5 TIPOS DE MENSAJE DE USUARIO

Tipos de mensaje	Respuestas correctas	Respuestas incorrectas	Respuestas incoherentes	Suma total de respuestas
Glucosa	19 (35,9%)	13 (24,5%)	21 (39,6%)	53
Insulina	10 (29,4%)	10 (29,6%)	14 (41,2%)	34
Ejercicio físico	16 (59,3%)	3 (11,1%)	8 (29,6%)	27
Comida	19 (51,4%)	10 (27%)	8 (21,6%)	37
Estrés	13 (50%)	3 (11,5%)	10 (38,5%)	26

y la frecuencia con la que se querría utilizar.

Evaluación 2: resultados del modelo de GLN

Un factor importante en la satisfacción del usuario al utilizar un agente conversacional es el tiempo de respuesta. Por ello, se ha decidido evaluar únicamente el modelo de 124 millones de parámetros de los dos que se entrenaron. Debido a las limitaciones de Dialogflow comentadas anteriormente, este proceso se ha llevado a cabo en la máquina local para no invertir más tiempo del necesario, pero se han utilizado copias de distintos mensajes reales escritos por los usuarios al conversar con el modelo. Estos mensajes tienen cinco temáticas alineadas con los *intents* para los cuales se ha activado las entregas: niveles de glucosa en sangre, dosis de insulina, ejercicio físico realizado, comida ingerida y una situación que causa estrés. Las respuestas generadas por el modelo a estos mensajes se han clasificado en tres tipos distintos: respuestas correctas, respuestas con contenido incorrecto que tienen coherencia en la conversación y respuestas incoherentes.

En la Tabla 1 se puede observar el número de mensajes de cada tipo con los que se ha probado el modelo y la clasificación de las respuestas obtenidas. Además del número, también se observa el porcentaje de respuestas de cada clase con respecto al total de respuestas de cada tipo de mensaje. Los tipos de mensaje que proporcionalmente han obtenido más respuestas correctas han sido los de ejercicio, comida y estrés, hallándose entre el 50 y 60%. Estas respuestas no necesariamente contenían información sobre la DM. Por el contrario, los mensajes con información sobre niveles de glucosa y dosis de insulina han sido los que menor porcentaje de respuestas correctas han obtenido, estando entre el 30 y 35%. En la mayoría de tipos de mensaje se han obtenido más respuestas incoherentes que respuestas incorrectas, cosa que indica que cuando el modelo genera una respuesta coherente tiene más probabilidad de ser correcta. Haciendo un análisis más a fondo de los mensajes con información de niveles de glucosa, se ha detectado que los mensajes que expresan el nivel de glucosa numéricamente obtienen un número de respuestas incorrectas o incoherentes más elevado que cuando lo expresan de forma cualitativa. En los mensajes con información sobre dosis de insulina también se han detectado resultados peores cuando se trata de insulina lenta con respecto a la insulina rápida.

VI. DISCUSIÓN

Si se analizan las valoraciones de cada usuario al agente conversacional utilizando la SUS, se puede concluir que se ha implementado una herramienta usable. Se ha observado que la única cuestión de la SUS que ha recibido una puntuación más baja en todos los usuarios ha sido la de la seguridad usando el sistema. Este resultado es comprensible debido a que las personas no se sienten seguras delegando decisiones importantes acerca de su salud a un agente conversacional y prefieren un humano experto en la materia. Pese a que el agente nunca llega a dar consejos sobre cómo regular la enfermedad, el hecho que pudiera llegar a hacerlo no parece inspirar confianza en los usuarios. Otra de las cuestiones peor puntuadas ha sido la que expresa la existencia de incoherencia en el sistema. Tras un análisis de los registros de conversaciones, se sospecha que esta valoración es debida a que ha habido algunos mensajes del usuario que el agente no ha asignado a ningún *intent*. Esto puede ser debido a que existan pocos *intents* o que falten frases de entrenamiento en algunos *intents*, lo que indica que es necesario mejorar la configuración del agente para que sea capaz de mantener conversaciones más correctas y variadas. La última de las cuestiones con peor puntuación ha sido la frecuencia con la que los usuarios querrían usar el agente conversacional. Se sospecha que los dos usuarios que han valorado esta cuestión peor tienen razones distintas para ello. El primero es una persona joven que ha expresado que las conversaciones se volvían repetitivas con el uso, lo que implica una reducción del uso de la herramienta con el tiempo. Esto es debido a que la herramienta se encuentra en una etapa inicial y gracias a estas evaluaciones se debería volver a una fase de planificación para añadir nuevas funcionalidades. La segunda usuaria es de edad más avanzada, y por tanto no está muy acostumbrada a utilizar este tipo de tecnología a diario. Esto conlleva que la intención de esta usuaria de utilizar el agente con frecuencia sea menor.

Al analizar los resultados obtenidos por el modelo de GLN, se observa que todavía hay bastante margen de mejora. Se ha obtenido un número de respuestas correctas más elevado para los mensajes con información que afecta en la gestión de la DM pero que no tiene por qué estar directamente relacionada. Esto es debido a que la primera base de datos que se ha utilizado para entrenar el modelo es mucho mayor a la segunda y contenía conversaciones en las cuales aparecen estos temas, pero sin la implicación de la DM. Al haber recibido mucho más entrenamiento de estas temáticas con respecto a los niveles de glucosa y las dosis de insulina, que únicamente aparecen en la segunda base de datos, es comprensible que los resultados sean mejores. El tipo de mensajes con un porcentaje de respuestas correctas mayor es el ejercicio físico con poco menos de un 60%. Este porcentaje tan bajo es comprensible por el poco entrenamiento que ha tenido el modelo, pero estos resultados deberían mejorar en un futuro al utilizar los historiales de conversaciones del agente para reentrenar el modelo.

VII. CONCLUSIONES

En este proyecto se ha cumplido el objetivo principal de crear un agente conversacional para el almacenamiento de información de diabetes mellitus (DM) tipo 1. Para ello, se ha realizado un amplio estudio de la bibliografía y herramientas disponibles para el desarrollo del agente y se han escogido Dialogflow para la configuración del agente y Firebase como base de datos. También se ha creado e implementado una metodología funcional para poder enviar respuestas dinámicas en la conversación elaboradas

por un modelo de generación de lenguaje natural (GLN), GPT-2. Sin embargo, no ha sido posible mostrar estas respuestas en la conversación con usuarios debido a una limitación de tiempo de respuesta de Dialogflow. Finalmente se ha evaluado la funcionalidad del modelo de GLN y la usabilidad del agente conversacional obteniendo resultados positivos para la usabilidad y mejorables para el modelo.

El siguiente paso para mejorar esta herramienta sería migrarla a un *framework* en que fuera posible extender el tiempo de respuesta permitiendo que el funcionamiento de las respuestas dinámicas generadas por el modelo fuera el esperado. El modelo se debería entrenar usando históricos de conversaciones para mejorar el porcentaje de respuestas correctas. También se debería mejorar la configuración del agente añadiendo más *intents* y frases de entrenamiento para los existentes. Otro campo en el que se podría profundizar sería este mismo tipo de desarrollo en un idioma distinto al inglés, como por ejemplo el español. Finalmente, sería ideal integrar el agente con más plataformas de mensajería para dotarlo de una mayor accesibilidad a usuarios, lo cual generaría más conversaciones y permitiría un mejor entrenamiento del modelo de GLN.

REFERENCIAS

[1] International Diabetes Federation. (2019). *Atlas de la Diabetes de la FID, Novena Edición.*

[2] American Diabetes Association. (2021). 2. Classification and diagnosis of diabetes: Standards of Medical Care in Diabetes. En *Diabetes Care* (págs. 44(Suppl. 1):S15 - S33).

[3] Yale, J.-F., Paty, B., & A., P. (2018). Hypoglycemia. En D. C. Committee, *Diabetes Canada 2018 Clinical Practice Guidelines for the Prevention and Management of Diabetes in Canada.* (págs. 42(Suppl 1):S88-S103).

[4] Zhou, M., Duan, N., Liu, S., & Shum, H.-Y. (2020). Progress in Neural NLP: Modeling, Learning, and Reasoning. *Engineering*(6), 275-290.

[5] Cho, K., van Merrienboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., & Schwenk, H. (2014). Learning phrase representations using RNN encoder–decoder for statistical machine translation. *arXiv:1406.1078*.

[6] Turchin, A., & Florez Builes, L. F. (2021). Using Natural Language Processing to Measure and Improve Quality of Diabetes Care: A Systematic Review. *Journal of Diabetes Science and Technology, 15(3)*, 553–560.

[7] Adamopoulou, E., & Moussiades, L. (2020). Chatbots: History, technology, and applications. *Machine Learning with Applications*(2).

[8] Chowdhury, G. G. (2003). Natural language processing. *Annual Review of Information Science and Technology, 37(1)*, 51-89.

[9] Kucherbaev, P., Bozzon, A., & Houben, G. J. (2018). Human-aided bots. *IEEE Internet Computing, 22(6)*, 36-43.

[10] Khanna, A., Pandey, B., Vashishta, K., Kalia, K., Bhale, P., & Das, T. (2015). A study of today's A.I. through chatbots and rediscovery of machine intelligence. *International Journal of U- and e-Service, Science and Technology, 8*, 277–284.

[11] Kim, J., Lee, H.-G., Kim, H., Lee, Y., & Kim, Y.-G. (2018). Two-step training and mixed encoding-decoding for implementing a generative chatbot with a small dialogue corpus. 31–35.

[12] Brown, B., Nasirzada, N., & Benson, D. (10 de Septiembre de 2021). *Botkit: Building Blocks for Building Bots*. Obtenido de https://github.com/howdyai/botkit/blob/main/packages/docs/index.md

[13] R. T. (2021). *Open source conversational AI*. Obtenido de RASA: https://rasa.com/

[14] G. C. (2021). *About ChatterBot*. Obtenido de ChatterBot 1.0.8 documentation: https://chatterbot.readthedocs.io/en/stable/#

[15] Microsoft. (2019). *Microsoft Bot Framework.* Obtenido de https://dev.botframework.com/

[16] Google. (2019). *Google assistant, your own personal google.* Obtenido de https://assistant.google.com/

[17] Microsoft. (2021). *LUIS (Language Understanding) - Cognitive Services - Microsoft*. Obtenido de https://www.luis.ai/home

[18] IBM. (2021). *IBM Watson*. Obtenido de https://www.ibm.com/watson

[19] Amazon. (2019). *Amazon Alexa Official Site: What is Alexa?* Obtenido de https://developer.amazon.com/es-ES/alexa

[20] Chaki, J., Ganesh, S. T., Cidham, S., & Theertan, S. A. (2020). Machine learning and artificial intelligence based Diabetes Mellitus detection and self-management: A systematic review. *Journal of King Saud University - Computer and Information Sciences*.

[21] Looije, R., Neerincx, M. A., & Cnossenc, F. (2010). Persuasive robotic assistant for health self-management of older adults: Design and evaluation of social behaviors. *Int. J. Human-Computer Studies 68*, 386–397.

[22] Sinoo, C., Pal, S. v., Henkemans, O. A., Keizer, A., Bierman, B. P., Looije, R., & Neerincx, M. A. (2018). Friendship with a robot: Children's perception of similarity between a robot's physical and virtual embodiment that supports diabetes self-management. *Patient Education and Counseling*, 1248–1255.

[23] Al-Taee, M. A., Al-Nuaimy, W., Muhsin, Z. J., & Al-Ataby, A. (2017). Robot Assistant in Management of Diabetes in Children Based on the Internet of Things. *IEEE INTERNET OF THINGS JOURNAL*.

[24] Magyar, G., Balsa, J., Cláudio, A., Carmo, M., Neves, P., Alves, P., . . . Guerreiro, M. (2019). Anthropomorphic Virtual Assistant to Support Self-care of Type 2 Diabetes in Older People: A Perspective on the Role of Artificial Intelligence. *Proceedings of the 14th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications*, 323-331.

[25] Spänig, S., Emberger-Klein, A., Sowa, J.-P., Canbay, A., Menrad, K., & Heider, D. (2019). The virtual doctor: An interactive clinical-decision-support system based on deep learning for non-invasive prediction of diabetes. *Artificial Intelligence In Medicine*.

[26] Radford, A., Wu, J., Child, R., Luan, D., & Amodei, D. (2019). Language Models are Unsupervised Multitask Learners.

[27] Woolf, M. (18 de Octubre de 2021). *minimaxir/gpt-2-simple: Python package to easily retrain OpenAI's GPT-2 text-generating model on new texts*. Obtenido de GitHub: https://github.com/minimaxir/gpt-2-simple

[28] AS, A. S. (Febrero de 2021). *Chatbot Dataset Topical Chat*. Obtenido de Kaggle: https://www.kaggle.com/arnavsharmaas/chatbot-dataset-topical-chat

[29] Pallets. (2010). *Welcome to Flask*. Obtenido de Flask Documentation Website: https://flask.palletsprojects.com/en/2.0.x/

[30] Brooke, J. (1996). SUS - A quick and dirty usability scale. *Usability evaluation in industry*.