Desarrollo de un chatbot de apoyo a la terapia basada en reminiscencia
A chatbot to support reminiscence therapy

Trabajo de Fin de Grado
Curso 2023–2024

Autor
Marta Vicente Navarro

Director
Gonzalo Mendez Pozo

Grado en Doble grado en Ingeniería Informática y Matemáticas

Facultad de Informática

Universidad Complutense de Madrid

Desarrollo de un chatbot de apoyo a la terapia basada en reminiscencia
A chatbot to support reminiscence therapy

Trabajo de Fin de Grado en Doble grado en Ingeniería Informática y Matemáticas

Autor
Marta Vicente Navarro

Director
Gonzalo Mendez Pozo

Convocatoria: *Junio* 2024

Grado en Doble grado en Ingeniería Informática y Matemáticas
Facultad de Informática
Universidad Complutense de Madrid

24 de mayo de 2024

Dedicatoria

A Mamá

Agradecimientos

Resumen

Desarrollo de un chatbot de apoyo a la terapia basada en reminiscencia

La enfermedad de Alzheimer es una condición neurodegenerativa que afecta las funciones cognitivas, la memoria, el pensamiento y el comportamiento. Aunque no es reversible, existen distintos tratamientos para mejorar la calidad de vida de quienes la padecen. Entre ellos destaca la terapia de reminiscencia. Este tratamiento no farmacológico se enfoca en evocar recuerdos haciendo que los pacientes ejerciten la mente y den un sentido de continuidad a su historia de vida.

Este proyecto tiene como objetivo desarrollar una herramienta que sirva de apoyo para la realización de estas terapias. En concreto, se centra en el desarrollo de un chatbot capaz de guiar el flujo de la conversación a lo largo de diferentes temas, extrayendo la información necesaria y generando preguntas especificas.

Con este propósito la primera etapa del trabajo se centra en la realización de un estudio exhaustivo que permita conocer cuáles son las mejores herramientas para su implementación. En concreto, se estudiarán las principales APIs y LLMs de procesamiento del lenguaje. A continuación, se desarrolla de forma iterativa e incremental el chatbot.

La herramienta final se basa en la API de Gemini, especialmente en el modelo "gemini-pro" para implementar las funciones de análisis de respuestas y generación de preguntas. Por otro lado, la interfaz desarrollada a través de Telegram permite un uso fácil y eficiente desde distintos dispositivos (móviles, tablet u ordenador).

El código asociado a este trabajo se puede consultar en el repositorio de GitHub.

Palabras clave

Reminiscencia, Chatbot, Historia de vida, Alzéhimer, API, Gemini, Procesamiento del lenguaje

Abstract

Desarrollo de un chatbot de apoyo a la terapia basada en reminiscencia

Alzheimer's disease is a neurodegenerative condition that affects cognitive functions, memory, thinking, and behavior. Although it is not reversible, there are various treatments to improve the quality of life for those affected. Among them, reminiscence therapy stands out. This non-pharmacological treatment focuses on evoking memories, allowing patients to exercise their minds and give continuity to their life story.

This project aims to develop a tool to support the implementation of these therapies. Specifically, it focuses on developing a chatbot capable of guiding the conversation flow across different topics, extracting the necessary information, and generating specific questions.

To this end, the first stage of the work focuses on conducting an exhaustive study to determine the best tools for its implementation. Specifically, the main language processing APIs and LLMs will be studied. Subsequently, the chatbot is developed iteratively and incrementally.

The final tool is based on the Gemini API, specifically the "gemini-pro" model to implement response analysis and question generation functions. Additionally, the interface developed through Telegram allows for easy and efficient use from various devices (mobile, tablet, or computer).

The code associated with this work can be found in the GitHub repository.

Keywords

Reminiscence, Chatbot, Life history, Alzheimer, API, Gemini, Language processing

Índice

Índice de figuras

Capítulo 1

Introducción

"Hay enfermos incurables, pero ninguno incuidable"
— Francesc Torralba

El constante aumento en la esperanza de vida a nivel mundial ha dado lugar a una población cada vez más envejecida. Este hecho ha traído consigo un aumento en las enfermedades asociadas con la vejez, como el Alzheimer y otros tipos de demencia. Los esfuerzos de la investigación en estos campos se han centrado en desarrollar tratamientos que puedan ralentizar el avance de esta enfermedad. La memoria juega un papel fundamental en la formación de nuestra identidad a través de nuestras experiencias y vivencias. Su pérdida puede ser devastadora, ya que afecta profundamente a la calidad de vida de la persona y de quienes la rodean.

Entre las estrategias no farmacológicas para el tratamiento del deterioro cognitivo y la memoria, destaca la terapia de reminiscencia. Esta técnica busca estimular los recuerdos pasados del paciente, lo que puede tener beneficios significativos para su bienestar social, mental y emocional.

Por otro lado, la inteligencia artificial está cada día más presente en nuestra vida diaria. Tiene una gran variedad de usos y aplicaciones que abarcan desde asistentes virtuales en nuestros teléfonos inteligentes hasta sistemas de recomendación en plataformas de streaming y comercio electrónico. Además, la inteligencia artificial se utiliza en la medicina para diagnósticos más precisos, en la industria para automatizar procesos de fabricación e incluso en la conducción autónoma de vehículos. Esta omnipresencia de la inteligencia artificial está transformando profundamente diversos sectores, impulsando la eficiencia, la personalización de servicios y generando nuevas oportunidades de desarrollo tecnológico.

El presente proyecto trata de encontrar la forma de aplicar la inteligencia artificial, y en concreto el procesamiento del lenguaje a la terapia de reminiscencia. En concreto, se busca desarrollar una herramienta conversacional de apoyo para la realización de este tipo de terapia.

Este capítulo tiene tres objetivos principales. En primer lugar, explicar cuál es la motivación actual con la que parte este problema. En segundo lugar, dejar claros cuáles son los objetivos que se plantean como punto de partida del proyecto.

Finalmente, con la intención de dar una visión global del proyecto, que permita entenderlo y hacerse una idea de como es la presente memoria, habrá una sección final en la que se explicará la estructura de este documento, haciendo hincapié en qué se puede encontrar en cada capítulo.

1.1. Motivación

A pesar de los avances en la investigación médica, todavía no se ha encontrado una cura definitiva para el Alzheimer. La mayoría de los tratamientos se centran en aliviar los síntomas y ralentizar la progresión de la enfermedad. Entre estos tratamientos, la terapia de reminiscencia ha surgido como una opción no farmacológica destacada.

Esta terapia se centra en estimular los recuerdos del pasado del paciente, lo que puede tener beneficios significativos en su bienestar social, mental y emocional. Ayuda a las personas a recordar y compartir experiencias pasadas, lo que puede ser reconfortante y estimulante, especialmente para aquellos que enfrentan el desafío de la pérdida de memoria asociada con el Alzheimer.

Durante el período académico 2023-2024, se desarrolló un *chatbot* con el objetivo de facilitar a los terapeutas la realización de terapias basadas en reminiscencia, simplificando y agilizando el proceso. Este desarrollo parte desde cero y busca crear un chatbot funcional y útil, capaz de mantener una conversación útil con el paciente para extraer la información necesaria.

Este Trabajo de Fin de Grado tiene como propósito asistir a terapeutas, familiares o amigos de pacientes con demencia en la obtención del material necesario para llevar a cabo terapias de reminiscencia. Se busca mejorar la eficacia de estas terapias y, en consecuencia, la calidad de vida tanto de los pacientes como de sus familiares.

1.1.1. Enfermedad de Alzheimer

La enfermedad de Alzheimer (EA) es la principal causa de demencia en adultos mayores. La EA es una enfermedad compleja que viene determinada por múltiples factores. En ocasiones es hereditaria. Está caracterizada por la pérdida de neuronas y sinapsis, junto con la presencia de placas amiloides y degeneración neurofibrilar. Clínicamente, se manifiesta como una demencia progresiva, comenzando con fallas en la memoria reciente y avanzando hasta la dependencia total del paciente.

Según Donoso (2003) La incidencia de la EA aumenta con la edad, siendo rara antes de los 50 años y afectando a un 1-2 % de los sujetos a los 60 años, 3-5 % a los 70, 15-20 % a los 80 y hasta la mitad de los mayores de 85 años. Es más frecuente en mujeres, posiblemente debido a la mayor longevidad.

La enfermedad se desarrolla en distintas etapas. En la etapa inicial, los síntomas pueden pasar desapercibidos o atribuirse a simples descuidos. La persona afectada puede experimentar dificultades para recordar nombres, eventos recientes o encon-

trar las palabras adecuadas en conversaciones. A pesar de estos desafíos, generalmente conservan la capacidad de realizar tareas cotidianas con cierta independencia. Sin embargo, es posible que comiencen a perder interés en actividades previamente disfrutadas.

A medida que la enfermedad progresa, los síntomas se vuelven más evidentes y problemáticos. La pérdida de memoria se vuelve más pronunciada, con dificultades para reconocer a familiares y amigos cercanos. Además, pueden surgir problemas de orientación en tiempo y espacio, lo que puede resultar en desorientación incluso en entornos familiares. Las habilidades de comunicación también se ven afectadas, con dificultades para seguir conversaciones o expresar pensamientos de manera coherente.

Finalmente, el Alzheimer alcanza su punto más devastador en la última etapa. La pérdida de memoria es profunda y completa, con una incapacidad para recordar incluso eventos recientes o reconocer caras familiares. La persona afectada puede experimentar cambios significativos en la personalidad y el comportamiento, volviéndose agitada, ansiosa o incluso agresiva en ocasiones. La capacidad para realizar actividades básicas de la vida diaria, como vestirse o alimentarse, se ve seriamente comprometida, y la supervisión constante se vuelve esencial.

El tratamiento farmacológico de la enfermedad de Alzheimer (EA) se basa en el uso de medicamentos para mejorar los defectos cognitivos y corregir trastornos conductuales. Los fármacos más valorados son los inhibidores de la acetilcolinesterasa, que intentan compensar la pérdida de neuronas colinérgicas en la corteza cerebral. El tratamiento también abarca fármacos para tratar trastornos conductuales, como antidepresivos, tranquilizantes o inductores del sueño (Chung y Cummings (2000)).

Además del tratamiento farmacológico, la actividad física y mental es fundamental para estimular la actividad cerebral y prevenir la pérdida de memoria. La orientación familiar es esencial para ayudar a los familiares a manejar los trastornos conductuales y aumentar la calidad de vida de los pacientes con EA. Un ejemplo de terapia no farmacológica que ayuda a mantener la actividad mental es la terapia de reminiscencia.

1.1.2. Terapias de reminiscencia

La reminiscencia, según la definición de la CEAFA (Confederación Española de Asociaciones de Familiares de personas con Alzheimer y otras demencias), es una técnica que busca evocar recuerdos en las personas, especialmente aquellos relacionados con eventos importantes de su vida.

En las terapias de reminiscencia se emplean diversos materiales, como fotografías, vídeos, noticias de periódicos, audios y objetos significativos, con el objetivo de estimular el recuerdo y activar la memoria a través de las emociones que despiertan estos elementos en el paciente.

Además de estimular los cinco sentidos para provocar el recuerdo, también se utiliza la narración de historias conocidas por el paciente. Por lo tanto, es crucial

conocer las experiencias pasadas del individuo afectado para adaptar los materiales utilizados en las terapias. La participación de personas cercanas al paciente, como familiares y cuidadores, en ejercicios de reminiscencia puede mejorar significativamente su calidad de vida.

Es recomendable construir relatos basados en las historias de vida del paciente para estos ejercicios de memoria. Las terapias de reminiscencia se consideran como parte de un proceso más amplio en el que el individuo intenta recuperar y vincular recuerdos que abarcan gran parte de su vida.

Una revisión sobre los posibles efectos de la terapia de reminiscencia en personas con demencia y sus cuidadores concluyó que disminuía las alteraciones de conducta y los síntomas depresivos, y que mejoraba la cognición (Huang et al. (2015)). En otra revisión se observaron mejorías en el estado de ánimo, la habilidad cognitiva, la conducta social y el bienestar general (Cotelli et al. (2012)), y recientemente se encontró que la terapia de reminiscencia mejoraba la salud física y que aumentaba la participación de los pacientes (Irazoki et al. (2017)).

1.1.3. Historia de vida

Las Historias de Vida son registros detallados de los aspectos más relevantes de la vida de un paciente, o de personas significativas para él. Son especialmente importantes en las primeras fases del Alzheimer, cuando la memoria aún es relativamente intacta. Estas historias proporcionan dignidad al paciente y permiten a quienes lo rodean conocer mejor su identidad, a medida que las pérdidas de memoria comienzan a ser significativas.

La creación de estas historias es vital para preservar la identidad del paciente. Pueden incluir detalles sobre la familia, la carrera profesional, aficiones, viajes y otros aspectos importantes de la vida del individuo. La elaboración puede realizarse de diversas formas, como escribir un libro, crear collages de fotos, producir una película o usar una "caja de memoria".

Es fundamental que estas historias reflejen la perspectiva personal del paciente, incluyendo emociones, sentimientos e interpretaciones. No se trata simplemente de relatar hechos cronológicos, sino de capturar la esencia única de la persona.

Un terapeuta suele encargarse de recopilar y estructurar los eventos importantes de la vida del paciente para crear la Historia de Vida. La estructura puede variar según las necesidades del paciente y los objetivos terapéuticos.

La participación de familiares o conocidos puede facilitar la evocación de recuerdos y enriquecer la Historia de Vida. Además, fortalece la comunicación entre el paciente y sus seres queridos, facilitando el proceso terapéutico.

1.2. Objetivos y metodología de trabajo

El objetivo de este trabajo es obtener un chatbot que sirva de apoyo para la realización de terapia de reminiscencia. En concreto, se pretende que el chatbot satisfaga los siguiente requisitos:

- Ser capaz de mantener una conversación, guiando el flujo de la misma a través de los temas predefinidos sobre los que se quiera conocer datos del paciente.
- A partir de las respuestas y la información que vaya dando el paciente, ser capaz de extraer la información buscada, identificar información adicional y ausente.
- Generar preguntas asociadas a las respuestas, de forma que se pueda obtener la información que en un primer momento hubiera faltado.
- Tener una interfaz usable y sencilla, que haga de este chatbot una herramienta útil.

Para cumplir con estos objetivos se seguirá la siguiente metodología:

- Estudiar la enfermedad del alzheimer y la terapia de reminiscencia para conocer qué características ha de cumplir una bot conversacional para servir de ayuda en este ámbito.
- Estudiar en profundidad las diferentes herramientas de procesamiento del lenguaje, desde bibliotecas como NLTK y spaCy, hasta las diferentes APIs y modelos LLM. Esto nos permitirá elegir las herramientas más apropiadas para llevar a cabo el desarrollo posterior.
- Una vez comenzada la fase de desarrollo, el primer paso será implementar un chatbot básico capaz de realizar preguntas predefinidas y almacenar las respuestas de forma eficiente.
- Mejorar la herramienta haciéndola más inteligente y capaz de analizar las respuestas, identificar la información omitida y hacer preguntas específicas para obtener la información faltante.
- Hacer una versión final capaz de analizar las repuestas y tirar del hilo, además de las utilidades que ya tenían los prototipos anteriores.
- Gestionar el almacenamiento de la información extraída.
- Generar una interfaz sencilla y usable que haga del chatbot una herramienta útil para el próposito para el que ha sido desarrollada.
- Permitir que el usuario pueda acompañar sus respuestas con imágenes. Por un lado, para ayudarle a dar respuestas más completas y por otro para obtener información adicional a través del procesamiento de las mismas.

Para llevar el control de versiones utilizaremos el repositorio de github: https://github.com/NILGroup/TFG-2324-ChatbotCANTOR.

1.3. Estructura de la memoria

El presente documento esta formado por 6 capítulos, incluyendo el actual. Estos se organizan como sigue.

- **Capítulo 1: Introducción**. El capítulo presente, se presenta el proyecto, motivación, objetivos y estructura del mismo.
- **Capítulo 2: Estado de la cuestión**. Es segundo capítulo, nos muestra una aproximación a las historias de vida, la enfermedad del Alzhéimer y la terapia de remiscencia. También muestra otros trabajos relacionados, contexto y precedentes del proyecto.
- **Capítulo 3: Tecnologías utilizadas** En este tercer capítulo se explica el estudio profundo de las herramientas de procesamiento del lenguaje, interfaces y almacenamiento de la información que se llevo a cabo para decidir cómo implementar el chatbot. Así se explican las diferentes opciones y los motivos que llevaron a elegir cada una de ellas.
- **Capítulo 4: Desarrollo de prototipos** Este capítulo cuenta cómo se ha ido desarrollando el proyecto, explicando cada una de las versiones que se han llevado a cabo. Se explica, desde la primera versión, los problemas que se han ido encontrando, las funcionalidades extra añadidas etc.
- **Capítulo 5: Diseño de un chatbot de ayuda a la terapia de reminiscencia** En el capítulo 5, se presenta la herramienta final con todas sus funcionalidades. El software y las herramientas utilizados, la guía para la instalación y puesta en marcha, la arquitectura del sistema final etc.
- **Capítulo 6: Conclusiones y trabajo futuro** El capítulo 6 trata acerca de las conclusiones a las que se ha llegado durante el desarrollo del proyecto, así como del trabajo que podría realizarse en el futuro para ampliarlo.

Introduction

"There are incurable patients, but none uncareable"
— Francesc Torralba

The constant increase in life expectancy worldwide has led to an increasingly aging population. This has brought with it an increase in diseases associated with old age, such as Alzheimer's and other types of dementia. Research efforts in these fields have focused on developing treatments that can slow the progression of this disease. Memory plays a fundamental role in shaping our identity through our experiences. Its loss can be devastating as it profoundly affects the quality of life of the person and those around them.

Among non-pharmacological strategies for treating cognitive decline and memory loss, reminiscence therapy stands out. This technique aims to stimulate the patient's past memories, which can have significant benefits for their social, mental, and emotional well-being.

On the other hand, artificial intelligence is increasingly present in our daily lives. It has a wide variety of uses and applications ranging from virtual assistants on our smartphones to recommendation systems on streaming platforms and e-commerce. Additionally, artificial intelligence is used in medicine for more accurate diagnoses, in industry to automate manufacturing processes, and even in autonomous vehicle driving. This omnipresence of artificial intelligence is profoundly transforming various sectors, driving efficiency, service personalization, and generating new technological development opportunities.

This project aims to find a way to apply artificial intelligence, specifically natural language processing, to reminiscence therapy. Specifically, it seeks to develop a conversational support tool for conducting this type of therapy.

This chapter has three main objectives. First, to explain the current motivation behind this problem. Second, to clarify the goals set as the project's starting point. Finally, with the intention of providing an overview of the project, there will be a final section explaining the structure of this document, emphasizing what can be found in each chapter.

1.4. Motivation

Despite advances in medical research, no definitive cure for Alzheimer's has yet been found. Most treatments focus on alleviating symptoms and slowing the progression of the disease. Among these treatments, reminiscence therapy has emerged as a prominent non-pharmacological option.

This therapy focuses on stimulating the patient's past memories, which can have significant benefits for their social, mental, and emotional well-being. It helps people recall and share past experiences, which can be comforting and stimulating, especially for those facing the challenge of memory loss associated with Alzheimer's.

During the 2023-2024 academic year, a chatbot was developed with the aim of facilitating therapists in conducting reminiscence-based therapies, simplifying and streamlining the process. This development starts from scratch and seeks to create a functional and useful chatbot capable of maintaining a useful conversation with the patient to extract the necessary information.

This Final Degree Project aims to assist therapists, family members, or friends of patients with dementia in obtaining the necessary material for conducting reminiscence therapies. It seeks to improve the effectiveness of these therapies and, consequently, the quality of life for both patients and their families.

1.4.1. Alzheimer's Disease

Alzheimer's disease (AD) is the leading cause of dementia in older adults. AD is a complex disease determined by multiple factors. Sometimes it is hereditary. It is characterized by the loss of neurons and synapses, along with the presence of amyloid plaques and neurofibrillary degeneration. Clinically, it manifests as progressive dementia, starting with recent memory failures and advancing to total dependency.

According to Donoso (2003), the incidence of AD increases with age, being rare before the age of 50 and affecting 1-2% of individuals at age 60, 3-5% at age 70, 15-20% at age 80, and up to half of those over 85. It is more frequent in women, possibly due to their greater longevity.

The disease develops in different stages. In the initial stage, symptoms may go unnoticed or be attributed to simple forgetfulness. The affected person may experience difficulties remembering names, recent events, or finding the right words in conversations. Despite these challenges, they generally retain the ability to perform daily tasks with some independence. However, they may begin to lose interest in previously enjoyed activities.

As the disease progresses, the symptoms become more evident and problematic. Memory loss becomes more pronounced, with difficulties recognizing close family and friends. Additionally, problems with time and space orientation may arise, leading to disorientation even in familiar environments. Communication skills are also affected, with difficulties following conversations or expressing thoughts coherently.

Finally, Alzheimer's reaches its most devastating stage. Memory loss is profound and complete, with an inability to recall even recent events or recognize familiar faces. The affected person may experience significant changes in personality and behavior, becoming agitated, anxious, or even aggressive at times. The ability to perform basic daily life activities, such as dressing or feeding, is severely compromised, and constant supervision becomes essential.

The pharmacological treatment of Alzheimer's disease (AD) is based on the use of medications to improve cognitive defects and correct behavioral disorders. The most valued drugs are acetylcholinesterase inhibitors, which attempt to compensate for the loss of cholinergic neurons in the cerebral cortex. Treatment also includes drugs to treat behavioral disorders, such as antidepressants, tranquilizers, or sleep inducers (Chung y Cummings (2000)).

In addition to pharmacological treatment, physical and mental activity is crucial for stimulating brain activity and preventing memory loss. Family guidance is essential to help family members manage behavioral disorders and increase the quality of life for AD patients. An example of a non-pharmacological therapy that helps maintain mental activity is reminiscence therapy.

1.4.2. Reminiscence Therapy

Reminiscence, according to the definition by CEAFA (Spanish Confederation of Associations of Relatives of People with Alzheimer's and Other Dementias), is a technique that seeks to evoke memories in people, especially those related to significant events in their lives.

Reminiscence therapies use various materials such as photographs, videos, newspaper clippings, audios, and significant objects to stimulate memory and activate it through the emotions these elements evoke in the patient.

In addition to stimulating the five senses to trigger memory, storytelling known to the patient is also used. Therefore, it is crucial to know the individual's past experiences to tailor the materials used in the therapies. The participation of people close to the patient, such as family members and caregivers, in reminiscence exercises can significantly improve their quality of life.

It is recommended to build stories based on the patient's life experiences for these memory exercises. Reminiscence therapies are considered part of a broader process in which the individual attempts to recover and link memories spanning much of their life.

A review on the potential effects of reminiscence therapy on people with dementia and their caregivers concluded that it decreased behavioral disturbances and depressive symptoms, and improved cognition (Huang et al. (2015)). Another review observed improvements in mood, cognitive ability, social behavior, and general well-being (Cotelli et al. (2012)), and recently, it was found that reminiscence therapy improved physical health and increased patient participation (Irazoki et al. (2017)).

1.4.3. Life Stories

Life Stories are detailed records of the most relevant aspects of a patient's life or significant people in their life. They are especially important in the early stages of Alzheimer's when memory is still relatively intact. These stories provide dignity to the patient and allow those around them to better understand their identity as memory losses become significant.

Creating these stories is vital for preserving the patient's identity. They can include details about family, professional career, hobbies, travels, and other important aspects of the individual's life. They can be created in various forms, such as writing a book, creating photo collages, producing a movie, or using a "memory box."

It is essential that these stories reflect the patient's personal perspective, including emotions, feelings, and interpretations. It is not simply about recounting chronological facts but capturing the unique essence of the person.

A therapist usually undertakes the task of collecting and structuring the significant events of the patient's life to create the Life Story. The structure can vary depending on the patient's needs and therapeutic goals.

The involvement of family members or acquaintances can facilitate memory recall and enrich the Life Story. Additionally, it strengthens communication between the patient and their loved ones, facilitating the therapeutic process.

1.5. Objectives and Work Methodology

The objective of this work is to develop a chatbot that supports the conduction of reminiscence therapy. Specifically, the chatbot should meet the following requirements:

- Be capable of maintaining a conversation, guiding the flow through predefined topics about which data from the patient is sought.
- Extract the desired information from the patient's responses, identify additional and missing information.
- Generate questions related to the responses, to obtain any initially missing information.
- Have a user-friendly interface, making this chatbot a useful tool.

To achieve these objectives, the following methodology will be followed:

- Study Alzheimer's disease and reminiscence therapy to understand the characteristics a conversational bot must have to be helpful in this area.

- Conduct an in-depth study of various natural language processing tools, from libraries like NLTK and spaCy to different APIs and LLM models. This will allow us to choose the most appropriate tools for subsequent development.
- Once the development phase begins, the first step will be to implement a basic chatbot capable of asking predefined questions and efficiently storing responses.
- Improve the tool, making it smarter and capable of analyzing responses, identifying omitted information, and asking specific questions to obtain the missing information.
- Develop a final version capable of analyzing responses and probing further, in addition to the functionalities of previous prototypes.
- Manage the storage of extracted information.
- Create a simple and usable interface, making the chatbot a useful tool for its intended purpose.
- Allow the user to accompany their responses with images, both to help them give more complete answers and to obtain additional information through processing them.

To manage version control, we will use the GitHub repository:
https://github.com/NILGroup/TFG-2324-ChatbotCANTOR.

1.6. Structure of the Document

This document consists of 6 chapters, including the current one. They are organized as follows:

- **Chapter 1: Introduction**. The current chapter presents the project, motivation, objectives, and structure.
- **Chapter 2: State of the Art**. This chapter provides an overview of life stories, Alzheimer's disease, and reminiscence therapy. It also presents related works, context, and precedents of the project.
- **Chapter 3: Technologies Used**. This third chapter explains the in-depth study of natural language processing tools, interfaces, and information storage conducted to decide how to implement the chatbot. It explains the different options and the reasons for choosing each.
- **Chapter 4: Prototype Development**. This chapter explains how the project has been developed, describing each version carried out. It details the initial version, the problems encountered, additional functionalities added, etc.

- **Chapter 5: Designing a Reminiscence Therapy Support Chatbot**. In Chapter 5, the final tool with all its functionalities is presented. The software and tools used, the guide for installation and setup, the architecture of the final system, etc.
- **Chapter 6: Conclusions and Future Work**. Chapter 6 discusses the conclusions reached during the project development and the work that could be done in the future to expand it.

Capítulo 2

Estado de la Cuestión

Este capítulo muestra la perspectiva teórica de la investigación llevada a cabo como trabajo previo al desarrollo del código. El conocimiento que se presenta en las siguientes páginas es necesario para entender cuál era el contexto del problema y el porqué de las decisiones que se han tomado para la construcción de la solución.

La estructura del capítulo muestra, en orden temporal, las herramientas del procesamiento de lenguaje que se han ido popularizando, desde las alternativas históricas hasta la situación actual. De está forma, nos permite entender su importancia y conocer otras posibles implementaciones.

Finalmente, este capítulo muestra trabajos relacionados, ya sean puntos de partida para el trabajo actual, trabajos que complementan al presente proyecto, o incluso otros trabajos que cuyo estudio ha servido como herramienta de aprendizaje de cara a preparar el *chatbot*.

En conclusión, se pretende dar una perspectiva global de la situación actual en la que se encuentra el procesamiento del lenguaje en general y el desarrollo del *chatbot* en particular.

2.1. Evolución del Procesamiento del Lenguaje Natural

El procesamiento del lenguaje natural (PLN) ha experimentado una evolución notable, impulsada por avances tecnológicos como la inteligencia artificial (IA). La integración de técnicas de IA en el PLN dió lugar a la rama del procesamiento natural del lenguaje (PLN), que supuso todo un hito en el ámbito del lenguaje escrito.

Inicialmente, los algoritmos de PLN se basaban en reglas, pero con el tiempo se adoptaron modelos de clasificación supervisada. Sin embargo, estos modelos enfrentaban limitaciones al no capturar el contexto completo de las palabras en una frase. Tres avances clave marcaron el camino hacia una PNL más avanzada.

Primero, el surgimiendo de los modelos *word embeddings* descritos en la sección 2.2, desarrollados en 2013 que permiten representar palabras en un espacio vectorial considerando su contexto, lo que facilita la comprensión de sinónimos y relaciones entre palabras.

En segundo lugar, la arquitectura de redes neuronales profundas conocida como *transformers*, descritos en la sección 2.3, revolucionó el campo al capturar el contexto completo de un texto mediante una matriz de atención. Además, los modelos basados en *transformers* permiten la transferencia de aprendizaje, lo que facilita la adaptación a diversas aplicaciones.

Finalmente, el surgimiento de modelos generativos de lenguaje multipropósito de gran tamaño, como el archiconocido ChatGPT, ha llevado la PNL a nuevos horizontes. Estos modelos, que se describen en la sección 2.4 con cientos de millones de parámetros, pueden generar texto de calidad comparable a la humana y están generando debates sobre su impacto en diversos ámbitos.

A pesar de estos avances, persisten desafíos en el PLN, como la necesidad de bases de datos específicas y validadas para el entrenamiento de modelos especializados. Los investigadores son llamados a trabajar en la construcción de nuevas bases de datos y en el desarrollo de esquemas de preprocesamiento y ajuste, con el objetivo de impulsar soluciones a problemas específicos a nivel global.

2.2. Word embeddings

Los *word embeddings* (Mikolov et al., 2023) son una técnica importante en el procesamiento del lenguaje natural que consiste en representar palabras y documentos como vectores numéricos en un espacio de dimensiones reales. Este enfoque permite que palabras con significados similares tengan representaciones vectoriales similares, lo que facilita que las computadoras comprendan el contenido basado en texto de manera más efectiva.

En los *word embeddings*, las palabras se representan como vectores numéricos en un espacio dimensional reducido, lo que permite capturar información semántica y sintáctica entre palabras. Estos vectores se utilizan como características para alimentar modelos de aprendizaje automático, lo que permite trabajar con datos de texto y preservar la información semántica y sintáctica. Existen diferentes técnicas para tratar los *word embeddings*.

2.2.1. TF-IDF

La frecuencia de término-inversa de frecuencia de documento (TF-IDF) es un algoritmo de aprendizaje automático que se utiliza para la incrustación de palabras en texto. Consta de dos métricas: frecuencia de término (TF) y la inversa de la frecuencia de documento (IDF).

Este algoritmo trabaja en una medida estadística para encontrar la relevancia

de las palabras en el texto, que puede estar en forma de un solo documento o varios documentos referidos como corpus.

$$\textbf{tf-idf}_{i,j} = \text{Frecuencia del término } i \text{ en el documento } j \times \text{Frecuencia inversa de documentos del término } i$$

El puntaje de frecuencia de término (TF) mide la frecuencia de las palabras en un documento particular. En otras palabras, cuenta la ocurrencia de palabras en los documentos.

$$\textbf{tf}_{i,j} = \frac{\textit{Frecuencia del término } i \textit{ en el documento } j}{\textit{Número total de términos en } j}$$

La inversa de la frecuencia de documento (IDF) mide la rareza de las palabras en el texto. Se le otorga más importancia que el puntaje de frecuencia de término porque, aunque el puntaje de TF otorga más peso a las palabras que ocurren con frecuencia, el puntaje de IDF se centra en las palabras raramente utilizadas en el corpus que pueden contener información significativa.

$$\textbf{idf}_i = \log\left(\frac{\textit{Número total de documentos}}{\textit{Número de documentos que contienen el término } i}\right)$$

El algoritmo TF-IDF se utiliza en tareas básicas de procesamiento de lenguaje natural y aprendizaje automático, como la recuperación de información, eliminación de palabras vacías, extracción de palabras clave y análisis de texto. Sin embargo, no captura eficientemente el significado semántico de las palabras en una secuencia.

2.2.2. Bolsa de palabras

El Bag of Words (BoW) es una técnica de procesamiento de texto ampliamente utilizada en el procesamiento del lenguaje natural (PLN) y la minería de texto. Esta técnica consiste en representar un documento de texto como un conjunto de palabras, ignorando el orden y la estructura gramatical.

En el modelo BoW, se crea un vocabulario de todas las palabras únicas en un conjunto de documentos de texto. Cada documento se representa como un vector de tamaño igual al tamaño del vocabulario, donde cada posición en el vector indica la frecuencia de una palabra en el documento.

Por ejemplo, si el vocabulario contiene las palabras "gato", "perroz "juguete", y un documento tiene una frecuencia de dos para "gatoz tres para "perro", su vector de representación BoW sería [2,3,0].

La técnica BoW es útil para la clasificación y agrupación de documentos basados en su contenido textual. Se utiliza en aplicaciones como análisis de sentimientos, clasificación de texto y recomendación de contenidos.

El proceso de creación de la matriz BoW se lleva a cabo en varios pasos:

1. **Creación del vocabulario**: Se genera un conjunto de palabras únicas a partir de todos los documentos de texto.
2. **Vectorización del texto**: Cada documento se convierte en un vector de tamaño igual al vocabulario, donde las palabras presentes en el documento tienen un valor de uno, y las ausentes tienen un valor de cero.
3. **Normalización del peso**: Los vectores se normalizan dividiendo cada valor por la suma total de valores en el vector.

Una vez creada la matriz BoW, se puede utilizar en diversas aplicaciones de inteligencia artificial, como la clasificación de textos, el análisis de sentimientos y la agrupación de documentos.

2.2.3. Word2Vec

El método Word2Vec, desarrollado por Google en 2013, revolucionó el campo del procesamiento del lenguaje natural (PLN) al ofrecer una forma innovadora de entrenar incrustaciones de palabras. Este enfoque se basa en una idea distributiva que utiliza skip-grams o una técnica llamada Bolsa Continua de Palabras (CBOW).

En esencia, Word2Vec emplea redes neuronales poco profundas con capas de entrada, salida y proyección. Estas redes tienen como objetivo reconstruir el contexto lingüístico de las palabras, considerando tanto su orden en el texto como su contexto futuro.

El proceso implica iterar sobre un corpus de texto para aprender las asociaciones entre palabras. Se fundamenta en la premisa de que las palabras que aparecen juntas en un texto tienen una similitud semántica entre ellas. Esto permite asignar representaciones vectoriales a las palabras que son cercanas geométricamente en el espacio vectorial.

La similitud del coseno se utiliza como métrica para medir cuán similares son dos palabras o documentos en su significado. Si el ángulo entre los vectores de palabras es pequeño, la similitud del coseno es alta, lo que indica que las palabras tienen significados similares. Por el contrario, si el ángulo es de 90 grados, la similitud es baja, lo que indica que las palabras son independientes en su contexto.

2.3. Transformers

Los modelos pre-entrenados son modelos de aprendizaje profundo o *DeepLearning* que sirven para realizar diversas tareas de Procesamiento de Lenguaje. Son pre-entrenados bajo grandes conjuntos de datos, lo que les permite ajustarse a atareas específicas sin requerir un entrenamiento desde cero. Este pre-entrenamiento es clave para entender por qué son tan valiosos, ya que permiten construir un sistema de generación de lenguaje sin un gran esfuerzo computacional (normalmente los entrenamientos tardan semanas o meses incluso con los mejores computadores).

Dentro de los modelos pre-entrenados, toman especial importancia los *transformers* (Vaswani et al. (2023)). Desde el primer momento, revolucionaron el PLN al introducir mecanismos de atención auto-ajustable, paralelización eficiente, captura de contexto bidireccional y pre-entrenamiento masivo. Todo ello permite construir modelos más poderosos y efectivos.

2.3.1. Aprendizaje por transferencia

Los modelos pre-entrenados, frente a los modelos estadísticos u otros tipos de algoritmos de *Deep Learning*, suponen grandes ventajas debido al pre-entrenamiento bajo un gran conjunto de datos y un pequeño ajuste posterior a realizar para adaptarlo a la tarea que se desee resolver, exigiendo mucho menos coste computacional y requiriendo para ello menos tiempo y esfuerzo. Todo esto es posible gracias al aprendizaje por transferencia o *Transfer Learning*.

El *Transfer Learning* es fundamental en el desarrollo y la eficiencia de los modelos pre-entrenados. Aparecen como solución al paradigma del aprendizaje aislado que presentan algoritmos como el aprendizaje supervisado tradicional. Estos modelos pueden aprovechar el conocimiento adquirido previamente en grandes conjuntos de datos para adaptarse a nuevas tareas con un costo computacional y de tiempo significativamente menor que si se entrenaran desde cero. Este enfoque es especialmente valioso en campos como la visión artificial o el PLN.

En el ámbito de la visión artificial, los modelos pre-entrenados pueden ser adaptados para tareas específicas, como la clasificación de imágenes, mediante el aprendizaje por transferencia. Por ejemplo en Tu et al. (2018), el sistema utiliza redes neuronales convolucionales pre-entrenadas para reconocer perros en imágenes, aprovechando el conocimiento previo adquirido por el modelo.

En PLN, los modelos pre-entrenados también son esenciales para resolver diversas tareas, como la detección de noticias falsas (Slovikovskaya (2019)). Estos modelos pueden emplear el aprendizaje por transferencia para adaptarse a nuevas tareas, como la clasificación de texto, utilizando el conocimiento previo obtenido durante el entrenamiento inicial.

Dentro del aprendizaje por transferencia, existen varias técnicas, como la adaptación de dominio, la confusión de dominio, el aprendizaje de una sola muestra ($One-shotLearning$), el aprendizaje de cero muestras ($Zero-shotLearning$) y el aprendizaje multitarea. Este último, utilizado por modelos como T5, tiene como objetivo crear modelos generalistas capaces de resolver múltiples tareas, en contraposición a modelos especializados en una sola tarea.

2.4. Modelos de Lenguaje de Gran Tamaño (LLM)

La invención de los transformadores marcó el comienzo de la era de los grandes modelos de lenguaje modernos. Desde 2018, los laboratorios de IA han comenzado

a entrenar modelos cada vez más grandes, conocidos como LLM.

Los Modelos de Lenguaje de Gran Tamaño (LLM) son modelos de aprendizaje profundo que se entrenan con grandes cantidades de datos utilizando la arquitectura de transformadores. Estos modelos constan de un codificador y un decodificador que trabajan en conjunto para entender y generar texto.

Hay tres lineas de desarrollo principales para estos modelos de lenguaje, y múltitud de modelos como se puee ver en la figura 2.1.

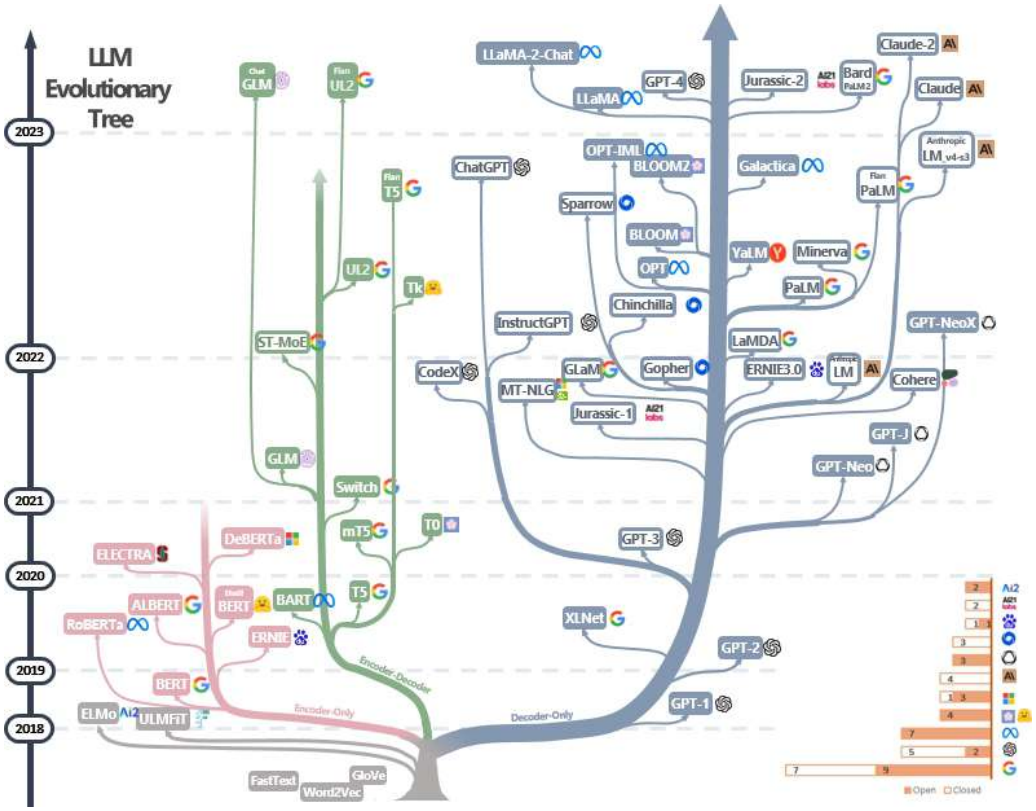

Figura 2.1: Desarrollo de los LLM porYang et al. (2023)

Por un lado, el grupo "solo codificador", mostrado en rosa en la figura 2.1 incluye LLM que son buenos para la comprensión del texto porque permiten que la información fluya en ambas direcciones del texto. En azul, podemos ver el grupo "solo decodificador" que incluye LLM que son buenos en la generación de texto porque la información solo fluye de izquierda a derecha del texto para generar nuevas palabras de manera eficiente y autorregresiva. Finalmente, hay un tipo codificador-decodificador (mostrado en verde) que combina ambos aspectos y se usa para tareas que requieren comprender una entrada y generar una salida, como la traducción.

Dentro de este último grupo encontramos la mayoría de los modelos que fueron considerados para el desarrollo de este trabajo como los distintos modelos de GPT, los de Google (Bard, que evolucionó a Gemini, LaMDA o PaLM).

2.4.1. BERT

Los modelos de lenguaje enmascarados, como los Masked Language Models, enmascaran un cierto porcentaje de palabras en una oración y se espera que el modelo prediga esas palabras en función del contexto restante Rothman (2022). Un ejemplo representativo de este enfoque es BERT.

BERT (*Bidirectional Encoder Representations from Transformers*) es un modelo de lenguaje enmascarado basado en la arquitectura Transformer Devlin et al. (2019). Desarrollado por Google en 2018, BERT ha demostrado un rendimiento sobresaliente en una variedad de tareas de procesamiento de lenguaje natural.

BERT se pre-entrena en dos grandes corpus de texto en inglés sin etiquetar: BookCorpus Zhu et al. (2015) y Wikipedia. Estos corpus garantizan una amplia cobertura de datos y una buena calidad para el entrenamiento del modelo.

Aunque inicialmente no estaba diseñado para la generación de texto, BERT ha sido adaptado para esta tarea, logrando mejoras significativas en comparación con modelos como GPT-2 Wang y Cho (2019).

Existen numerosas variantes de BERT, algunas pre-entrenadas en dominios específicos y otras con un ajuste fino para tareas específicas Rajasekharan (2019). Por ejemplo, Beto es la versión en español de BERT, entrenada en un corpus extenso en dicho idioma Cañete et al. (2020).

La innovación clave de BERT es su enfoque bidireccional, que permite al modelo comprender el contexto de una palabra en función de su entorno completo. A diferencia de modelos unidireccionales como GPT-2, BERT considera tanto el contexto anterior como el posterior a una palabra dada.

La arquitectura de BERT consiste en una pila de codificadores de transformadores, con un número variable de capas dependiendo de la versión del modelo. Este enfoque permite a BERT realizar múltiples tareas de procesamiento de lenguaje, incluyendo Modelado de Lenguaje Enmascarado y Predicción de la Siguiente Oración Devlin et al. (2019).

2.4.2. T5

T5, o Text-to-Text Transfer Transformer Raffel et al. (2020), es un marco unificado para el aprendizaje por transferencia en procesamiento de lenguaje natural (PLN). Este enfoque revolucionario convierte todos los problemas basados en texto en un formato de texto a texto, donde el modelo recibe texto como entrada y genera nuevo texto como salida. Inspirado en marcos anteriores para tareas de NLP, como la modelización de lenguaje o la extracción de fragmentos, T5 permite aplicar el mismo modelo, objetivo de entrenamiento, procedimiento de entrenamiento y proceso de decodificación a cada tarea considerada.

El objetivo principal de T5 es proporcionar una perspectiva exhaustiva sobre el estado actual del campo de la transferencia de aprendizaje para PLN. En lugar de

proponer nuevos métodos, se enfoca en la exploración y comparación empírica de técnicas existentes. Además, para facilitar futuros trabajos en este campo, se liberan conjuntos de datos, modelos pre-entrenados y código fuente.

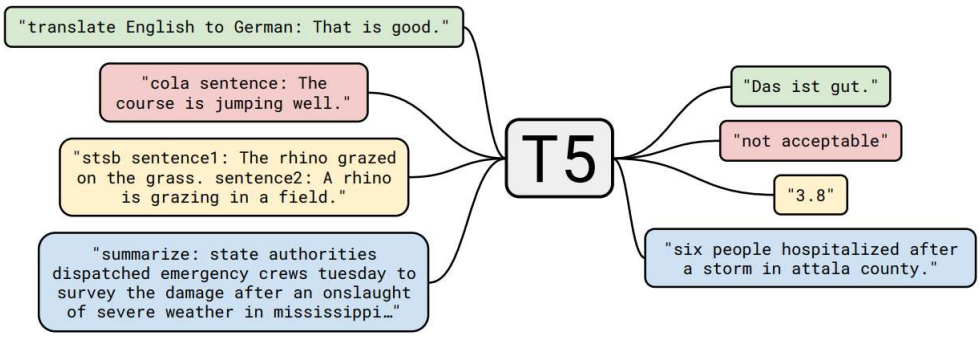

Figura 2.2: Diagrama del modelo T5 de Raffel et al. (2020)

Cómo se puede ver en la figura 2.2 el modelo T5 nos permite realizar diversas tareas como la traducción o la realización de resumenes.

2.4.3. GPT (Generative Pretrained Transformer)

Los modelos de lenguaje informales, también conocidos como modelos de lenguaje generativos, se centran en predecir palabras o tokens enmascarados dentro de una oración. Visualicemos un token enmascarado como un espacio en blanco dentro de una frase. En este escenario, el modelo solo considera el contexto anterior (palabras a la izquierda) y no tiene en cuenta el contexto posterior. La dirección del procesamiento no es crucial, siempre y cuando siga una dirección única, utilizando sólo las palabras relevantes del contexto para hacer predicciones, mientras descarta el resto. Esta propiedad clave de los modelos se refleja en su esquema de entrenamiento (Rothman, 2022).

2.4.3.1. GPT-1

GPT-1 (Generative Pre-trained Transformer 1) fue el primer gran modelo de lenguaje desarrollado por OpenAI tras la introducción de la arquitectura Transformer por Google en 2017. En junio de 2018, OpenAI publicó un artículo titulado "Improving Language Understanding by Generative Pre-Training" (Radford et al., 2018), donde presentaron este modelo inicial junto con el concepto general de un Transformer generativo pre-entrenado.

Hasta ese momento, los modelos de procesamiento de lenguaje neuronal con mejor desempeño se basaban principalmente en el aprendizaje supervisado a partir de grandes conjuntos de datos etiquetados manualmente. Esta dependencia limitaba su utilidad en conjuntos de datos poco etiquetados y hacía que el entrenamiento de modelos extremadamente grandes fuera costoso y lento. Muchos idiomas (co-

mo el suajili o el criollo haitiano) resultaban difíciles de abordar debido a la escasez de datos disponibles para la construcción de corpus. En contraste, el enfoque "semi-supervisado" de GPT implicaba dos etapas: una etapa de pre-entrenamiento generativo no supervisado, donde se establecían los parámetros iniciales mediante un objetivo de modelado de lenguaje, y una etapa de ajuste fino discriminatorio supervisado, donde estos parámetros se adaptaban a una tarea específica.

La elección de la arquitectura Transformer, en lugar de técnicas anteriores basadas en redes neuronales con atención mejorada, dotó a los modelos GPT de una memoria más estructurada que la alcanzada mediante mecanismos recurrentes, lo que resultó en un sólido desempeño en transferencia a diversas tareas.

2.4.3.2. GPT-2

GPT-2 es uno de los modelos más emblemáticos de lenguaje casual que sigue la arquitectura Transformer basada en auto-atención (masked self-attention). Presentado por OpenAI en 2019, fue aclamado desde el principio en el campo del Procesamiento de Lenguaje debido a su gran escala, con más de 1.5 billones de parámetros.

El objetivo principal de este sistema es construir una distribución de probabilidad donde cada posible palabra a generar recibe una probabilidad en función del contexto anterior. Este modelo fue pre-entrenado en un gran corpus de texto inglés utilizando el método auto-supervisado (self-supervised). Su propósito fundamental es la predicción de la siguiente palabra en una secuencia de palabras u oraciones.

Para el pre-entrenamiento, se creó un conjunto de datos llamado WebText, obtenido al extraer millones de páginas web a partir de enlaces de salida de Reddit que cumplían con ciertos criterios de calidad. Las páginas de Wikipedia asociadas a estos enlaces fueron excluidas. El resultado fue un corpus masivo de 40GB de textos adaptados para el entrenamiento de este modelo (Radford et al., 2019).

La estructura de GPT-2 se asemeja a la del Transformer original. Inicialmente, el modelo constaba de un codificador y un decodificador diseñados para tareas específicas como traducción automática. Sin embargo, GPT-2 abandonó esta arquitectura convencional y optó por una serie exclusiva de decodificadores basados en el Transformer. El número de decodificadores utilizados varía según el tamaño de GPT-2: desde doce en la versión Small hasta cuarenta y ocho en la versión Extra Large. Esta configuración basada únicamente en decodificadores es una característica distintiva de GPT-2, ilustrada en la Figura 2.3.

2.4.3.3. GPT-3

GPT-3 aumentó significativamente la cantidad de parámetros en comparación con GPT-2, alcanzando hasta 175 mil millones, lo que permitió una mayor complejidad y capacidad de aprendizaje. Gracias a su mayor capacidad y tamaño de conjunto de datos, GPT-3 demostró una mejor habilidad para generalizar y comprender una variedad más amplia de contextos y tareas. Además, requiere menos entrenamiento adicional para tareas específicas en comparación con GPT-2.

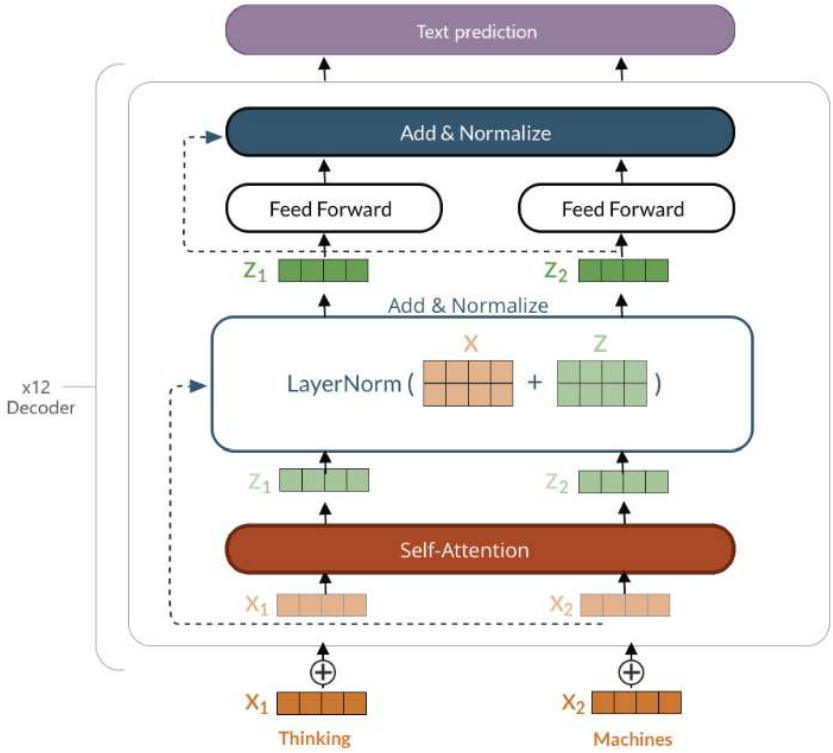

Figura 2.3: Arquitectura de GPT-2

Por otro lado, a pesar de su mayor capacidad, GPT-3 logró reducir la generación de texto tóxico en comparación con GPT-2, aunque todavía se necesitaron estrategias de mitigación.

GPT-3 produce textos con mayor precisión y coherencia, lo que resulta en una calidad general de generación de texto más alta y una capacidad para realizar tareas más complejas.

2.4.3.4. GPT-4

GPT-4,(Achiam et al., 2023) es el siguiente modelo de gran escala de OpenAI capaz de aceptar entradas de imagen y texto para producir salidas de texto. Aunque menos capaz que los humanos en muchos escenarios del mundo real, GPT-4 exhibe un rendimiento a nivel humano en diversos puntos de referencia profesionales y académicos, incluida la aprobación de un examen simulado de abogacía con una puntuación aproximadamente un 10 % superior a la media de los examinados humanos. GPT-4 es un modelo basado en Transformers preentrenado para predecir el siguiente token en un documento. El proceso de ajuste posterior al entrenamiento mejora su rendimiento en medidas de factualidad y adherencia al comportamiento deseado.

En una serie de benchmarks tradicionales de Procesamiento de Lenguaje Natural (NLP), GPT-4 supera tanto a modelos de lenguaje grandes anteriores como a

la mayoría de los sistemas de vanguardia (que a menudo requieren entrenamiento específico para el benchmark o ingeniería manual). En el benchmark MMLU, que cubre 57 temas en inglés, GPT-4 no solo supera a los modelos existentes en inglés, sino que también demuestra un rendimiento sólido en otros idiomas.

A pesar de sus capacidades, GPT-4 tiene limitaciones similares a modelos anteriores, como la falta de fiabilidad completa, una ventana de contexto limitada y la incapacidad de aprender de la experiencia. Las capacidades y limitaciones de GPT-4 plantean desafíos de seguridad significativos y novedosos..

2.4.4. Llama

A diferencia de la creencia común de que más parámetros conducen a un mejor rendimiento, investigaciones recientes muestran que modelos más pequeños entrenados con más datos pueden superar a los modelos más grandes. Se ha desarrollado una serie de modelos llamados LLaMA, Touvron et al. (2023) que van desde 7B hasta 65B de parámetros, con un rendimiento competitivo en comparación con los mejores LLMs existentes.

Por ejemplo, LLaMA-13B supera a GPT-3 en la mayoría de las pruebas, a pesar de ser 10 veces más pequeño. Se espera que estos modelos democratizen el acceso y el estudio de los LLMs, ya que pueden ejecutarse en una sola GPU. Además, se asegura la compatibilidad con la fuente abierta al utilizar solo datos públicamente disponibles, a diferencia de otros modelos que dependen de datos no disponibles públicamente. El trabajo detalla las modificaciones realizadas en la arquitectura del $transformers$ 2.3 y el método de entrenamiento, además de presentar el rendimiento de los modelos en comparación con otros LLMs en una serie de pruebas estándar. También se examinan los sesgos y la toxicidad codificados en los modelos, utilizando benchmarks de la comunidad de inteligencia artificial responsable.

El conjunto de datos de entrenamiento es una mezcla de varias fuentes, que cubren un conjunto diverso de dominios. Mayormente reutiliza fuentes de datos que se han utilizado para entrenar otros LLMs, con la restricción de utilizar solo datos públicamente disponibles y compatibles con la distribución abierta.

La tokenización de los datos se lleva a cabo con el algoritmo de codificación de bytes (BPE), utilizando la implementación de $SentencePiece$ [1] . Dividimos todos los números en dígitos individuales y recurrimos a bytes para descomponer caracteres UTF-8 desconocidos. El tamaño total de nuestro conjunto de datos de entrenamiento contiene aproximadamente 1.4T de tokens después de la tokenización. La mayoría de los datos de entrenamiento se utilizan solo una vez durante el entrenamiento, con la excepción de los dominios de Wikipedia y Libros, sobre los cuales se realizan aproximadamente en dos épocas.

[1] El algoritmo $SentencePiece$ Kudo y Richardson (2018) utiliza un enfoque basado en subpalabras, donde construye un vocabulario de subpalabras que se adaptan a la frecuencia de aparición en el corpus de entrenamiento.

2.4.5. Modelos LLM de Google AI

A lo largo de los años Google AI ha desarrollado varios modelos LLM, todos ellos basados en Transformers, aunque con diferentes alcances y capacidades. A continuación se presentan estos modelos, aunque se explicaran con más profundidad a lo largo de la memoria.

2.4.5.1. LaMDA

LaMDA (Modelo de Lenguaje para Aplicaciones de Diálogo) fue el primer modelo LLM de Google AI. Su primera iteración fue anunciada durante la conferencia Google I/O de 2021, donde se presentó como un modelo conversacional de lenguaje. La segunda versión, presentada al año siguiente, introdujo mejoras y nuevas capacidades, como la generación de conversaciones originales sobre temas no previamente enseñados.

La atención sobre LaMDA aumentó cuando el ingeniero de Google, Blake Lemoine, afirmó que el chatbot se había vuelto sensible, lo que generó debates sobre la efectividad de la prueba de Turing para evaluar la inteligencia artificial general.

A diferencia de la mayoría de los modelos de lenguaje, LaMDA fue entrenado específicamente en diálogo. Durante su entrenamiento, adquirió sutilezas que distinguen las conversaciones abiertas de otras formas de lenguaje, como el sentido común.

La arquitectura Transformer utilizada por LaMDA es un modelo de solo decodificador, pre-entrenado en un corpus que incluye documentos y diálogos. Ha sido ajustado y probado con diferentes configuraciones de hiperparámetros, demostrando superar las respuestas humanas en ciertas áreas específicas.

2.4.5.2. Bard

Bard, anunciado por Google AI en 2022, se presenta como la evolución natural de su predecesor, LaMDA. Si bien este último ya destacaba por su capacidad para generar texto, traducir idiomas, escribir contenido creativo y responder a preguntas de forma informativa, Bard va un paso más allá.

Bard se nutre de un conjunto de datos de texto y código aún más extenso que LaMDA, lo que le permite acceder y procesar información del mundo real a través de la Búsqueda de Google. Esto se traduce en respuestas más completas, precisas y actualizadas, ya que Bard se mantiene al día con los últimos acontecimientos y datos disponibles en la web. Su potencial creativo se expande a la escritura de diferentes tipos de contenido, desde poemas y código hasta guiones, piezas musicales, correos electrónicos y cartas.

Además de la arquitectura Transformer, Bard incorpora otras técnicas de vanguardia como la atención y la decodificación autoregresiva. La atención permite a Bard centrarse en las partes más relevantes de un texto, mientras que la decodifica-

ción autoregresiva le permite generar texto palabra por palabra de forma coherente y fluida.

2.4.5.3. Gemini

Gemini es la última generación de modelos de lenguaje grandes de Google AI. Se anunció en 2024 y se basa en la arquitectura de Bard. Gemini es capaz de realizar todas las tareas que Bard puede hacer, y además tiene algunas características nuevas, como la capacidad de generar diferentes formatos de texto creativo, como poemas, código, guiones, piezas musicales, correo electrónico, cartas, etc.

Su razonamiento multimodal avanzado le permite comprender y responder a preguntas complejas que involucran diferentes tipos de información, desde datos textuales hasta imágenes y vídeos. Gemini mejora constantemente sus habilidades y rendimiento a medida que se expone a nuevos datos y experiencias.

2.5. Alucinaciones

En el campo de la inteligencia artificial (IA), se utiliza el término "alucinación" o "alucinación artificiaL" para describir respuestas de IA que no parecen estar justificadas por los datos de entrenamiento Edwards (2023). Este fenómeno, también conocido como confabulación o delirio, se refiere a la generación de respuestas o creencias que carecen de base sólida Ji et al. (2022). Por ejemplo, un chatbot alucinado podría ofrecer información falsa, como afirmar que los ingresos de Tesla fueron de 13.600 millones de dólares, un número aparentemente inventado Lin (2022). Otros ejemplos de alucinaciones serían la generación respuestas inventadas como se puede ver en la imagen 2.4, donde ChatGPT genera un resumen de un artículo que no existe, o la adición de elementos en el estudio de imágenes, como se ve en la figura2.5.

Figura 2.4: Ejemplo de alucinación de ChatGPT Wikipedia (Fecha de última modificación)

Figura 2.5: Ejemplo de alucinación en una imagen Rohrbach et al. (2023)

Es importante destacar que, aunque se usa el término "alucinación" por analogía con la psicología humana, la diferencia fundamental radica en que las alucinaciones de IA se relacionan con respuestas injustificadas más que con percepciones falsas. Algunos investigadores señalan que este término antropomorfiza de manera poco razonable a los ordenadores.

El problema de las alucinaciones de IA se volvió prominente alrededor de 2022 con la introducción de modelos grandes de lenguaje como ChatGPT (Zhuo et al., 2023). Los usuarios expresaron preocupación por la capacidad de estos bots para insertar falsedades aparentemente plausibles en sus respuestas. En 2023, los analistas reconocieron las alucinaciones frecuentes como un desafío significativo en la tecnología de modelos de lenguaje (Leswing, 2023).

Se cuestiona la confiabilidad del contenido generado por inteligencia artificial en el ámbito científico (Machin-Mastromatteo, 2023). Según Spinak (2023), los modelos de lenguaje de IA pueden percibir patrones que son imperceptibles para los humanos, lo que resulta en resultados inesperados o incorrectos, fenómeno conocido como "alucinación" (Spinak, 2023). Estas alucinaciones pueden llevar a la producción de contenido falso, especialmente fuera de sus dominios específicos o al tratar con temas complejos o ambiguos, lo cual puede ser lingüísticamente plausible pero no científicamente preciso (Sage, 2023).

2.6. Estado actual del procesamiento del lenguaje

En los últimos años, el avance del procesamiento del lenguaje ha sido notable, con numerosas ventajas, aunque también con riesgos asociados al desarrollo de software.

Por un lado, los modelos de inteligencia artificial evolucionan rápidamente, generando frecuentes actualizaciones etiquetadas como "última versión estable". Esta denominación se refiere a la versión más reciente del modelo que ha sido probada y validada para su uso generalizado, ofreciendo calidad y fiabilidad.

Por otro lado, la IAs se enfrentan al problema de la regulación. La presidenta de la Comisión Europea, Ursula von der Leyen, ha subrayado que la IA está trans-

formando nuestras vidas y ha enfatizado la importancia de un enfoque sensato y generalizado para beneficiar a la economía y la sociedad (Comisión Europea, 2023). La recién aprobada Ley de IA de la UE, considerada como el primer marco global en esta materia, busca promover la innovación responsable al regular los riesgos identificados y garantizar la seguridad y los derechos fundamentales de las personas y las empresas.

Este enfoque se basa en evaluar el riesgo de los sistemas de IA: aquellos de bajo riesgo, como los sistemas de recomendación, tendrán libertad y ninguna obligación, mientras que los de alto riesgo, como infraestructuras críticas, sistemas de salud y aplicaciones policiales, deberán cumplir requisitos estrictos de mitigación, calidad de datos y supervisión humana. Además, se prohibirán los sistemas de IA que representen una amenaza clara para los derechos fundamentales, como la manipulación del comportamiento humano. Se introducirán normas específicas para garantizar la transparencia en los modelos de IA de uso general, junto con multas para las empresas que no cumplan con las regulaciones.

En diciembre de 2023, se elaboró un borrador sobre la regulación de la IA por parte del Consejo de la Unión Europea y el Parlamento Europeo. Finalmente, el 2 de febrero de 2024, se aprobó la primera ley del mundo sobre inteligencia artificial: la Ley de IA. Después de una reunión en Bruselas, los embajadores de los 27 Estados miembros dieron su visto bueno político a esta normativa, tras la presentación en enero de la versión final del texto y la creación de la Oficina Europea de Inteligencia Artificial. Aunque algunos países mostraron oposición hasta el último momento, el camino continuó su curso y se prevé que entre 2024 y 2030 todos los países adopten la Ley de IA (ElDerecho.com, 11-04-2024).

Los constantes cambios en los modelos y las modificaciones en las legislaciones nacionales suponen un desafío para los desarrolladores de software. Durante el desarrollo de este proyecto, nos enfrentamos a diversas situaciones derivadas de estos sucesos. Por un lado, tuvimos que buscar una alternativa a la API de Bard, que se transformó en Gemini entre diciembre de 2023 y febrero de 2024, y por otro lado, hacer frente a las limitaciones impuestas por las diferentes leyes, que se solvento mediante el uso de VPN.

De hecho, la situación cambia tan rápidamente, que en la recta final del proyecto hemos observado varios cambios en los términos y condiciones de uso de la API de Gemini a nivel mundial. De hecho, la próxima actualización de los términos de uso será el 22 de mayo de 2024.

En conclusión, en el campo del desarrollo del lenguaje y la extracción de información a partir de imágenes, la IA está experimentando numerosos avances en los últimos años. Sin embargo, es fundamental tener en cuenta que el uso de los distintos modelos está sujeto a cambios tanto en las versiones y sus características, como en las leyes y términos y condiciones de uso.

2.7. Otros trabajos relacionados

2.7.1. Proyecto Cantor

El proyecto CANTOR (Composición automática de narrativas personales como apoyo a terapia ocupacional basada en reminiscencia) en el que se enmarca este trabajo, desarrolla herramientas digitales utilizando tecnologías de Inteligencia Artificial para construir automáticamente historias de vida que puedan ser reexaminadas posteriormente como apoyo a las terapias ocupacionales de pacientes con demencias.

CANTOR está financiado por el Ministerio de Ciencia e Innovación, en colaboración entre académicos de la Universidad Complutense de Madrid y la Universidad de La Coruña. El objetivo de CANTOR es desarrollar herramientas que faciliten la terapia ocupacional basada en reminiscencia para mejorar la calidad de vida de pacientes con deterioro cognitivo.

En este ámbito se han elaborado varios Trabajos de Fin de Grado en la Facultad de Informática de la UCM. Paso a referir algunos de ellos relacionados con este trabajo.

2.7.1.1. Generación de historias de vida usando técnias de Deep Learning

En el curso 2021-2022, la compañera María Cristina Alameda Salas (Salas, 2020), en su trabajo de fin de grado, Generación de historias de vida usando técnicas de Deep Learning, desarrolló un sistema basados en técnicas de Deep Learning que de soporte a la generación de historias de vida. Partiendo de unos datos de entrada en forma de datos estructurados de tipo biográfico, ese trabajo permite la construcción de un sistema de generación de lenguaje natural, transformador de los datos de entrada a un escrito fluido y coherente, que abarque la representación de los datos de partida de manera completa, sin incorrecciones y lo más cercana posible a una redacción humana. Nuestro objetivo ahora sería el desarrollo de un programa que interactuara con el usuario y nos permitiera obtener toda esa información biográfica que da lugar a las historias de vida.

2.7.1.2. Extracción de preguntas a partir de imágenes para personas con problemas de memoria mediante técnicas de Deep Learning

En 2021, en la UCM se desarrollo un trabajo de extracción de preguntas a partir de imagenes con técnicas de Deep Learning (Boto et al., 2021). Este proyecto ayuda a las personas con problemas de memoria a recordar aspectos de su vida utilizando técnicas de IA como redes convolucionales y recurrentes. Para lograrlo se desarrolló un sistema capaz de extraer preguntas de fotografías que puedan representar recuerdos para las personas con problemas de memoria utilizando un bot que simula una sesión de terapia de reminiscencia. El usuario ha de enviar fotografías al bot y este se encargará de enviarle, una a una, las preguntas generadas por la red neuronal.

En este momento, el usuario deberá recordar todo lo posible sobre la imagen para poder responder a las preguntas y conseguir ejercitar su memoria.

2.7.1.3. Extracción de información personal a partir de redes sociales para la creación de un libro de vida

Este proyecto, desarrollado por Aguilera Heredero y Molina Muñoz (2021), tiene como objetivo principal ayudar a terapeutas ocupacionales en el tratamiento de pacientes con problemas de memoria, especialmente aquellos relacionados con el deterioro cognitivo asociado con la edad. Se propone la creación de una herramienta para desarrollar un libro de vida.

La herramienta combina técnicas de extracción y tratamiento de datos de diferentes redes sociales proporcionadas por el paciente, almacenándolos en una base de datos SQL para obtener la información más relevante. Esta información se utilizará para crear el libro de vida, que se presentará en una interfaz web desarrollada con React. La interfaz permitirá visualizar fácilmente los datos recopilados, utilizando tablas, mapas, líneas de tiempo y galerías de fotos.

2.7.1.4. Generación de historias a partir de una base de conocimiento

En este proyecto, desarrollado por Magaz (2023), se construyó una aplicación para crear relaciones entre palabras e imágenes, partiendo de unas palabras determinadas. Con las relaciones establecidas la aplicación genera estadísticas a través de las cuales puede evaluarse el progreso del paciente. En cada sesión se trata un tema concreto, pudiéndose elegir el tipo de sesión entre Sesión palabras (se elige una categoría de palabras), Sesión Progreso (se visualiza el avance del paciente a través de estadísticas agrupadas por categorías) o Sesión imágenes (donde se asocia una imagen a un concepto y una categoría).

2.7.1.5. Recuerdame 1.0

Recuerdame 1.0 (Barquilla Blanco et al., 2022) presenta la creación de una aplicación que facilite a los terapeutas la realización de terapias basadas en reminiscencia para tratar a pacientes con alzheimer, haciéndolas más ágiles y rápidas.

La aplicación creada es una aplicación web responsive, con una estructura Modelo Vista Controlador creada mediante lenguajes como HTML, CSS, PHP, JavaScript. La aplicación tiene una usabilidad aceptable, pero tiene detalles que mejorar y algunas funcionalidades que no pudieron ser desarrolladas por la falta de tiempo.

2.7.1.6. Recuerdame 2.0

Durante el curso 2022-2023, la aplicación recuerdame 1.0 fue mejorada dando lugar a recuerdame 2.0 (Díez Sobrino et al., 2023), para ofrecer una experiencia

terapéutica más enriquecedora a pacientes con problemas de memoria. Las mejoras incluyeron la optimización basada en la retroalimentación de usuarios finales, la narración mejorada de Historias de Vida, la generación automática de resúmenes, la integración de terapia con un bot, la mensajería entre terapeutas y cuidadores, y la capacidad de generar vídeos de Historias de Vida. Además, se realizó una evaluación exhaustiva del sistema en instituciones médicas y residencias de ancianos para validar su eficacia.

2.7.2. Celia

Celia, [2] es un chatbot impulsado por inteligencia artificial (IA) desarrollado por la compañía Atlantic, con el respaldo de la Xunta de Galicia en España. Este chatbot tiene como objetivo acompañar, entretener y brindar asistencia a las personas mayores y dependientes, y se destaca por su capacidad para detectar indicios y patrones de enfermedades neurodegenerativas, como el Alzheimer, mediante el análisis de la voz del usuario.

A diferencia de otros asistentes de conversación como Alexa o Siri, Celia va más allá al utilizar herramientas biométricas para medir y monitorear parámetros indicativos no solo de enfermedades neurológicas, sino también de condiciones emocionales como la ansiedad y la depresión.

Celia está disponible para su uso a través de tres plataformas: WhatsApp, la versión web y una aplicación oficial disponible actualmente solo para dispositivos Android. Los usuarios pueden interactuar con Celia a través de mensajes de texto o de voz, y la instalación es sencilla, lo que permite un acceso rápido y eficiente a este recurso tecnológico.

Una característica destacada de Celia es su capacidad para tomar la iniciativa en las conversaciones y proponer actividades sin necesidad de instrucciones. Además, ofrece la posibilidad de establecer recordatorios para citas médicas o la toma de medicamentos, brindando un apoyo integral en la gestión de la salud de los usuarios.

[2] Así es Celia, la inteligencia artificial para adultos mayores que puede detectar indicios de alzheimer

Capítulo 3

Tecnologías utilizadas

El capítulo actual detalla las tecnologías empleadas en la construcción y el despliegue del chatbot diseñado para asistir en terapias de reminiscencia. También se analizarán las herramientas y metodologías utilizadas en el proceso de desarrollo de prototipos, que se presenta en el capítulo 4, así como su integración con los conceptos y conocimientos presentados en el estado de la cuestión.

En cada sección del capítulo, se llevará a cabo un análisis de las diferentes alternativas consideradas para la construcción de cada uno de los módulos que componen el chatbot. Desde la evaluación de diversas API's y bibliotecas de Procesamiento del Lenguaje Natural (PLN) hasta la exploración de las múltiples opciones disponibles para el desarrollo de la interfaz de usuario.

Esta metodología permitirá proporcionar un contexto completo y entendible que facilitará la comprensión de las decisiones tomadas a lo largo del desarrollo del proyecto, como se detalla en el capítulo 4.

3.1. Bibliotecas de Procesamiento del Lenguaje en Python

3.1.1. NLTK

La biblioteca NLTK (Natural Language Toolkit) [1] ofrece una amplia gama de herramientas y recursos para tareas de PLN.

En primer lugar, NLTK permite realizar tareas como la tokenización y el etiquetado POS (Part-Of-Speech tagging). Al utilizar las herramientas de tokenización, podemos dividir el texto en unidades más pequeñas, lo que facilita el análisis y la comprensión. El etiquetado POS asigna etiquetas gramaticales a cada palabra en el texto, lo que nos permite identificar la función de cada palabra en la oración.

[1] Página oficial de NLTK

```
sentence = "Reminiscence Therapy involves the
   discussion of past activities using prompts like
   photos."
tokens = nltk.word_tokenize(sentence)
tagged = nltk.pos_tag(tokens)
tagged[0:len(tagged)]
```

Listing 3.1: Ejemplo de código en Python. Se pueden consultar más ejemplos en Bird et al. (2009)

En este código *nltk* tokeniza la oración introducida y etiqueta cada *token* indicando la categoría sintáctica de cada *token* como sigue:

- NNP: Nombre propio singular
- NN: Nombre, singular o sustantivo singular
- VBZ: Verbo tercera persona del singular presente
- DT: Determinante
- IN: Preposición o oración subordinada
- JJ: Adjetivo
- NNS: Nombre plural
- VBG: Verbo, gerundio o participio
- . : Signo de puntuación

```
>>>[('Reminiscence', 'NNP'), ('Therapy','NNP'), ('
   involves','VBZ'), ('the', 'DT'), ('discussion', 'NN
   '), ('of', 'IN'), ('past', 'JJ'), ('activities', '
   NNS'), ('using', 'VBG'), ('prompts', 'NNS'), ('like
   ', 'IN'), ('photos', 'NNS'), ('.', '.')]
```

Listing 3.2: Tokenización y etiquetado con nltk

Otras de las funcionalidades que nos permite esta biblioteca es el análisis sintáctico o lematización. Por ejemplo, nos permite la obtención de árboles sintácticos, lo que permite visualizar la estructura gramatical de las oraciones, facilita el análisis y la interpretación del texto.

```
entities = nltk.chunk.ne_chunk(tagged)
nltk.download('treebank')
from nltk.corpus import treebank
t = treebank.parsed_sents('wsj_0001.mrg')[0]
```

Listing 3.3: Análisis sintáctico y lematización con nltk

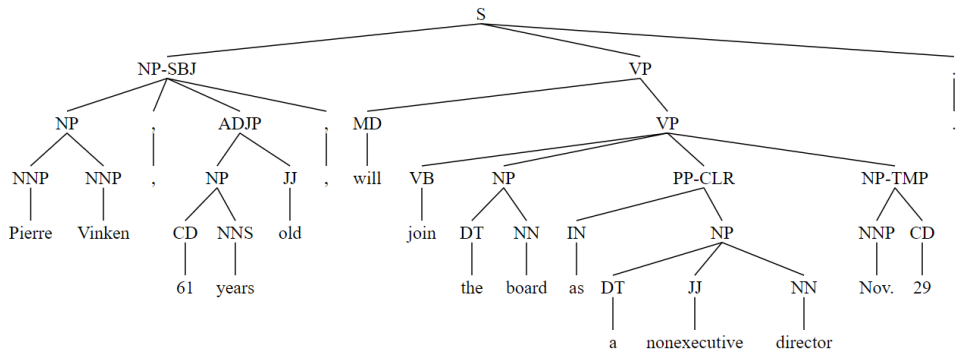

Figura 3.1: Árbol sintáctico generado con nltk

Además de estas características fundamentales, NLTK ofrece una serie de otras funcionalidades que amplían aún más su utilidad. Por ejemplo, incluye herramientas para la extracción de entidades nombradas, el análisis de sentimientos, la generación de texto y la traducción automática. Estas capacidades adicionales permiten abordar una amplia variedad de tareas en el procesamiento del lenguaje natural, desde la clasificación de texto hasta la generación de resúmenes automáticos y la traducción de idiomas. En resumen, NLTK es una herramienta invaluable para investigadores, estudiantes y profesionales que trabajan en el campo del PLN, ofreciendo una amplia gama de funcionalidades que facilitan el análisis, la comprensión y la manipulación del lenguaje humano.

3.1.2. SpaCy

SpaCy [2]ofrece soporte para más de 25 idiomas y cuenta con 84 pipelines de entrenamiento. Utiliza el aprendizaje multi-tarea con modelos preentrenados como BERT, lo que permite un rendimiento avanzado en tareas de procesamiento del lenguaje natural. Sus componentes incluyen herramientas para el reconocimiento de entidades nombradas, etiquetado de partes del discurso, análisis de dependencias, segmentación de oraciones, clasificación de texto, lematización, análisis morfológico, vinculación de entidades y más.

```
import spaCy

# Load English tokenizer, tagger, parser and NER
nlp = spacy.load("en_core_web_sm")

# Process whole documents
text = ("Reminiscence Therapy involves the discussion
   of past activities using prompts like photos.")
doc = nlp(text)

# Analyze syntax
```

[2]spaCy

```
print("Noun phrases:", [chunk.text for chunk in doc.
    noun_chunks])
print("Verbs:", [token.lemma_ for token in doc if
    token.pos_ == "VERB"])
```

Listing 3.4: Ejemplo de tokenización usando spaCy

Este código carga el modelo preentrenado "*en_core_web_sm*" de spaCy, que incluye herramientas para tokenizar, etiquetar, analizar la sintaxis y reconocer entidades nombradas en textos en inglés. Luego, procesa el texto proporcionado y muestra las frases nominales identificadas utilizando la función *noun_chunks* y los verbos lematizados utilizando la propiedad *lemma_*. Este análisis gramatical permite identificar las partes clave del texto, como los sustantivos y las acciones descritas.

```
>>> Noun phrases: ['Reminiscence Therapy', 'the
    discussion', 'past activities', 'prompts', 'photos'
    ]
Verbs: ['involve', 'use']
```

Listing 3.5: Resultado de tokenización usando spaCy

Además, es fácilmente ampliable con componentes y atributos personalizados, y es compatible con modelos personalizados en PyTorch, TensorFlow y otros frameworks. Spacy ofrece visualizadores integrados para la sintaxis y el reconocimiento de entidades nombradas, y facilita el empaquetado, despliegue y gestión de flujos de trabajo de modelos. Con su precisión rigurosamente evaluada y su robustez, Spacy es una herramienta poderosa y versátil para el procesamiento del lenguaje natural.

La biblioteca spaCy cuenta con diferentes componentes que interactúan entre sí esuchando la salida unos de otros para mejorar su procesamiento (*listener*). Además, existen una serie de reglas y dependencias entre los componentes. Por ejemplo, el módulo *atribute_ruler* indica proporciona reglas de etiquetado a *tagger*.

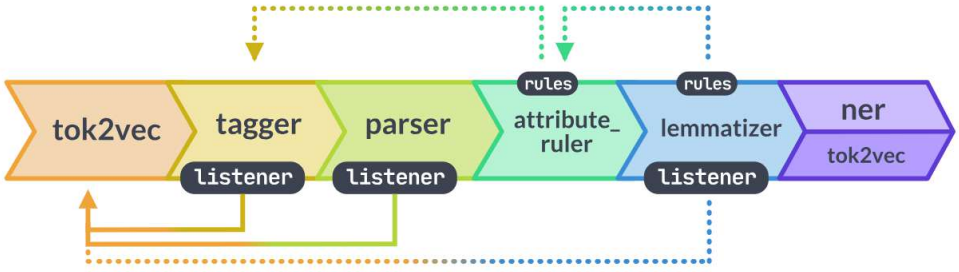

Figura 3.2: Árbol sintáctico generado con nltk

El gráfico representa la estructura de una tubería de procesamiento de lenguaje natural (NLP) en spaCy, mostrando la secuencia de componentes y sus interacciones.

- tok2vec: Este componente convierte los tokens en vectores de palabras, que capturan el significado semántico de las palabras en el contexto de la oración.

- tagger: El *tagger* asigna etiquetas gramaticales a cada token en el texto, como partes del discurso (POS).
- parser: El analizador sintáctico analiza la estructura sintáctica del texto, identificando las relaciones de dependencia entre las palabras.
- *attribute_ruler*: Este componente aplica reglas para agregar atributos adicionales a los tokens, como excepciones de lema y POS, y manejar espacios en blanco de manera coherente.
- *lemmatizer*: El lematizador determina la forma base de cada palabra (su lema) en función de su contexto y su parte del discurso.
- *ner/tok2vec*: El componente de reconocimiento de entidades (NER) identifica entidades nombradas en el texto, como nombres de personas, lugares o organizaciones. En algunos modelos, este componente comparte la representación de vectores de palabras (tok2vec) con otros componentes para mejorar la coherencia y la precisión de las predicciones.

En resumen, este gráfico muestra cómo los componentes de spaCy interactúan entre sí para realizar tareas de procesamiento de lenguaje natural, aprovechando la información compartida y las reglas definidas para mejorar la precisión y la coherencia del análisis lingüístico.

3.2. APIs de procesamiento del lenguaje

Las APIs de procesamiento del lenguaje son conjuntos de herramientas y servicios que integran múltiples funcionalidades relacionadas con el PLN en sus aplicaciones y sistemas. Es decir, son herramientas que aúnan y ofrecen funcionalidades como el análisis de sentimientos, el reconocimiento de entidades o la tokenización. Frente a las bibliotecas y modelos presentados anteriormente tienen la ventaja de que se pueden usar sin necesidad de desarrollar desde cero algoritmos o modelos, lo que facilita su uso. Estas características hacen que este tipo de APIs sean comúnmente usadas en variedad de aplicaciones, desde chatbots hasta sistemas de recomendación.

3.2.1. Bard

Bard es una API de procesamiento del lenguaje natural desarrollada por Google con el objetivo de ofrecer respuestas conversacionales coherentes y relevantes a través de interacciones de mensajes. Basada en LaMDA, un modelo de lenguaje experimental de Google, Bard compite directamente con ChatGPT en el campo del procesamiento del lenguaje natural, permitiendo realizar consultas y recibir respuestas sin necesidad de navegar por diferentes páginas web.

Google ha priorizado la accesibilidad y la transparencia en el desarrollo de Bard, ofreciendo modelos y recursos de PLN de código abierto que pueden ser utilizados

y modificados por la comunidad. Inicialmente lanzado con un modelo reducido de LaMDA 2.4.5.1, Bard busca ampliar su alcance y obtener comentarios para su mejora continua.

```
prompt = "A partir del texto a continuación, que contiene información
 sobre una persona y damelo en una lista info donde
 info[nombre]:valor_atributo."

texto = "Hola mi nombre es Marta, tengo 22 años y soy de Zaragoza"
>> **Info[nombre:Marta;edad:22;ciudad:Zaragoza]**
¿Hay algo más que pueda hacer por ti?
```

Como se puede ver en el ejemplo, Bard es capaz de analizar la información proporcionada y generar respuestas coherentes y formateadas según las especificaciones dadas. Gracias a su capacidad para comprender el contexto y generar texto de manera precisa, Bard es una herramienta valiosa para tareas que requieren PLN, como la generación de respuestas conversacionales. Todo ello lo convierte en una gran opción para una amplia gama de aplicaciones, desde chatbots hasta sistemas de asistencia virtual.

Sin embargo, Bard ya no está disponible. En diciembre de 2023, Google fortaleció la capacidad de Bard al incorporar Gemini Pro en inglés, brindando habilidades más avanzadas de comprensión, razonamiento, resumen y codificación. Más adelante, en febrero de 2024, se anunció la expansión de Gemini Pro a más de 40 idiomas y se oficializó el cambio de nombre de Bard a Gemini, lo que implicó descartar el primer modelo del proyecto desarrollado en Bard debido a la indisponibilidad de Gemini en España en ese momento.

3.2.2. Gemma

Gemma es una API de PLN desarrollada por OpenAI. Utiliza modelos de lenguaje basados en la arquitectura GPT (Generative Pre-trained Transformer) para una variedad de tareas de PLN, como producción de texto, análisis de sentimientos, clasificación de texto, y más. Gemma se destaca por su capacidad para generar texto coherente y de alta calidad en una variedad de estilos y tonos, así como por su facilidad de uso y su API intuitiva.

Una de las principales características de Gemma es su rendimiento en tareas de generación de texto, donde ha establecido nuevos estándares de calidad y coherencia en muchos casos. Además, Gemma ofrece modelos pre-entrenados en varios dominios y lenguas, lo que facilita su integración en una variedad de aplicaciones de PLN. Sin embargo, debido a su enfoque en modelos de última generación, Gemma puede requerir recursos computacionales significativos y puede ser más difícil de entender y utilizar para usuarios principiantes en PLN.

En primer lugar, estos modelos se trabajaron en Google Collaborate aumentando el número de GPUs. De esta forma Gemma tiene un buen comportamiento y produce

respuestas adecuadas y coherentes. En concreto, una respuesta generada por *gemma-7b* sería la siguiente.

```
prompt =gemma_lm.generate("What is the meaning of life?",
 max_lenght = 64)
 >>> The question is one of the most important questions in the world.
  It's the question that has been asked by philosophers, theologians and
  scientist for centuries. And it's the question that has been asked by
  people who are looking for answer to their own lives.
```

Sin embargo, las limitaciones propias de Google Collaborate no permitían en la versión gratuita aumentar el número de GPUs de forma frecuente y en consecuencia tuve que estudiar el modelo en otro entorno. Para ejecutarlo de forma local y obtener un buen comportamiento es necesario instalar Linux y descargar el modelo de Hugging face.

Aunque *gemma-2b* en la versión local instalada desde Hugging Face en Linux tiene un buen comportamiento, genera respuestas incoherentes que hacen de este modelo poco útil para nuestro proyecto. La versión *gemma-7b* genera respuestas mucho mejores pero tiene la enorme desventaja de que ocupa una gran cantidad de espacio en memoria.

3.2.3. GPT API

GPT API es una API de PLN desarrollada por OpenAI. Utiliza modelos de lenguaje basados en la arquitectura GPT (Generative Pre-trained Transformer) para una variedad de tareas de PLN, como generación de texto, análisis de sentimientos y clasificación de texto entre otras. GPT API se caracteriza por generar texto lógico, coherente y de calidad en múltiples estilos y tonos, así como por su facilidad de uso y su API intuitiva.

Una de las principales propiedades de GPT API es su rendimiento en tareas de generación de texto, donde ha instaurado nuevos estándares de calidad en muchos casos. Además, GPT API ofrece también modelos pre-entrenados en varios dominios y lenguas, simplificando su integración en diversas tareas de PLN. Por otro lado, al enfocarse en modelos de última generación, GPT API puede requerir recursos computacionales importantes y es más difícil de entender y utilizar para usuarios principiantes en PLN.

Sin embargo para obtener el comportamiento que se necesitaba en este trabajo debía ser entrenada, y debido a las limitaciones hardware esto suponía una cantidad de tiempo inviable.

3.2.4. Rasa

Entre las opciones que se barajaron para seguir desarrollando el proyecto se encuentra Rasa.

Rasa es una plataforma de código abierto diseñada para el desarrollo de chatbots y asistentes virtuales conversacionales. Utilizando técnicas de procesamiento de lenguaje natural y aprendizaje automático, Rasa permite a los desarrolladores crear sistemas de diálogo inteligentes y personalizados.

Una de las principales ventajas de Rasa es su flexibilidad y personalización, ya que los desarrolladores tienen control total sobre el comportamiento y la lógica de sus chatbots. Además, Rasa proporciona herramientas robustas para la gestión del diálogo, la comprensión del lenguaje natural y la integración con otros sistemas. Sin embargo, una posible desventaja de Rasa es su curva de aprendizaje, ya que requiere un conocimiento sólido de PLN y aprendizaje automático para aprovechar al máximo su potencial. Además, debido a su naturaleza de código abierto, puede requerir más tiempo y recursos para implementar y mantener en comparación con otras soluciones comerciales. Hay que señalar que, aunque rasa no es la API más potente en cuánto a generación de texto, tiene numerosas aplicaciones que resultan interesantes.

3.2.5. Gemini

Gemini es una API de procesamiento del lenguaje natural (PLN) desarrollada por Google que permite a los usuarios interactuar con modelos de lenguaje avanzados para generar texto coherente y relevante en respuesta a consultas y solicitudes. Utiliza modelos de lenguaje de última generación entrenados por Google, que son capaces de comprender y generar texto en varios idiomas y contextos. Los usuarios pueden enviar texto de entrada a través de la API y recibir respuestas generadas por los modelos de Gemini. Ofrece varios modelos para satisfacer diferentes necesidades y casos de uso, entre los que se encuentran:

- **gemini-pro**: Optimizado para entradas de texto.
- **gemini-pro-vision**: Optimizado para entradas multimodales de texto e imágenes.

Gemini puede utilizarse para una variedad de aplicaciones, incluyendo generación de texto a partir de entradas bien sean de texto, o imágenes, conversaciones de varios turnos (chats) o para la obtención de embeddings para modelos del lenguaje.

3.2.5.1. Procesamiento de texto

El modelo más apropiado para el procesamiento de texto es $gemini-pro$. Es una herramienta avanzada diseñada para el procesamiento eficiente de texto, ofreciendo una amplia gama de funcionalidades que facilitan el análisis y manipulación de datos textuales. A continuación se muestran algunas de sus funcionalidades.

- Análisis de sentimientos y emociones: Ofrece capacidades para analizar sentimientos en el texto, identificando tonos positivos, negativos o neutros, útiles para el análisis de opiniones o comentarios.

```
prompt = "Identifica los sentimientos en el siguiente texto: "
 + "Me encanta este producto, es increíble."
>>> Sentimiento: Positivo
```

- Extracción de información: Puede extraer automáticamente información específica como nombres, fechas o lugares y además hacerlo en un formato específico. Esta funcionalidad será especialmente útil en este trabajo.

```
prompt = "Extrae información en formato json de la siguiente
 frase: La reunión será el 15 de mayo en Barcelona."

>>>{"fecha": "15 de mayo", "ubicación": "Barcelona"}
```

- Generación de resúmenes y extractos: Permite generar resúmenes automáticos o extractos de textos largos, facilitando la comprensión rápida del contenido principal. Esta funcionalidad también nos será de utilidad para la generación de historias de vida.

```
prompt = "Resume : Este artículo presenta un estudio exhaustivo
sobre inteligencia artificial ."

>>>"El artículo aborda el tema de la inteligencia artificial en
 detalle."
```

- Clasificación y categorización: La herramienta puede clasificar textos en categorías predefinidas o identificar temas principales utilizando algoritmos de aprendizaje automático.

```
prompt = "Clasifica el siguiente texto" + "El mercado de
criptomonedas experimenta un crecimiento significativo."

>>>Categoría: Finanzas
```

3.2.5.2. Procesamiento de imágenes

Los mensajes multimodales son instrucciones que utilizan modelos avanzados de lenguaje y admiten diferentes tipos de entradas, como texto e imágenes. Al combinar múltiples formatos de entrada, estos mensajes permiten una amplia gama de aplicaciones, como la clasificación de imágenes, el reconocimiento de escritura a mano, la traducción y otras tareas creativas.

En esta sección, exploraremos los tipos de instrucciones que pueden generarse al ingresar tanto imágenes como texto en el modelo Gemini. A través de ejemplos interesantes, veremos cómo este modelo puede procesar esta información multimodal y producir respuestas en forma de texto.

En la actualidad, Gemini [3] tiene la capacidad de procesar solicitudes que combinan texto e imagen como entrada, y luego generar respuestas basadas únicamente en texto. El texto puede utilizarse para contextualizar la imagen o para solicitar al modelo que opere y genere una respuesta relacionada con la imagen.

Un ejemplo muy simple que destaca la capacidad del LLM de reconocer la existencia de algo en una imagen o no y de responder al desarrollador de manera booleana, es el que muestra la figura 3.3. Este enfoque puede ser útil en la detección de contenido específico.

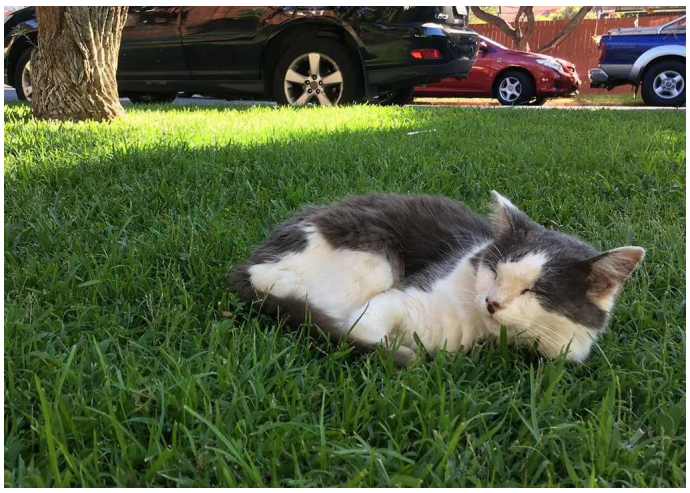

Figura 3.3: Imagen de entrada a Gemini en el primer ejemplo

```
>>> ¿En esta imagen aparece un gato?
Respuesta de Gemini:  True
```

Otro ejemplo es el asociado a la figura, 3.4. Aquí vemos el poder de Gemini para narrar historias y formas más creativas de usar la IA generativa.

```
>>> Escribe un haiku sobre esta foto
Respuesta de Gemini:

Un banco junto al lago,

Una vista de las montañas más allá,

Un momento de paz.
```

Finalmente, un ejemplo en el que se combinan varias habilidades de Gemini es el que se muestra en la figura 3.5. No solo reconoce las formas, sino que también entiende que, si bien las formas son levemente dibujadas, están destinadas a ser formas distintas vinculadas matemáticamente con atributos específicos (p.ej., 3 lados, 4 lados, 5 lados).

[3] Página oficial de Google con más ejemplos

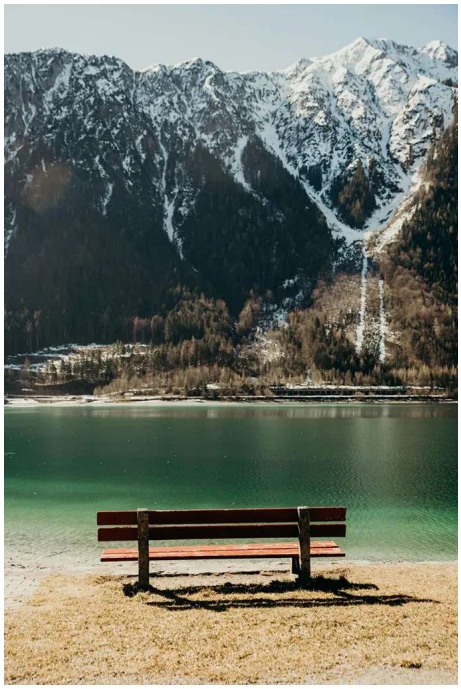

Figura 3.4: Imagen de entrada a Gemini en el segundo ejemplo

Además, la presencia del signo de interrogación no confunde a Gémini en su interpretación de la progresión lógica de las formas geométricas. En cambio, Gemini identifica que esta es una progresión matemática de 3, 4 a 5 y que, por lo tanto, la última forma tendría 6 lados y propone un hexágono.

```
>>> ¿Qué sigue? Explica tu razonamiento
Respuesta de Gemini: El triángulo tiene 3 lados, el cuadrado tiene 4 lados
 y el pentágono tiene 5 lados. El número de lados aumenta en 1 para cada
  forma. Por lo tanto, la siguiente forma debería tener 6 lados, que es un
   hexágono.
```

3.3. Almacenamiento de la información

3.3.1. JSON

El JSON, acrónimo de JavaScript Object Notation, es un formato ligero para estructurar datos que se asemeja a los mapas en la programación. Está diseñado para representar datos de manera legible para las máquinas y fácilmente interpretable por los humanos. Se utiliza ampliamente en el intercambio de datos entre aplicaciones web y en el manejo de respuestas de API. El formato JSON consta de pares de clave-valor, donde las claves son únicas y los valores pueden ser cadenas, booleanos, números, objetos JSON o matrices JSON.

En Python, el formato de datos más cercano a JSON es el diccionario. El módulo

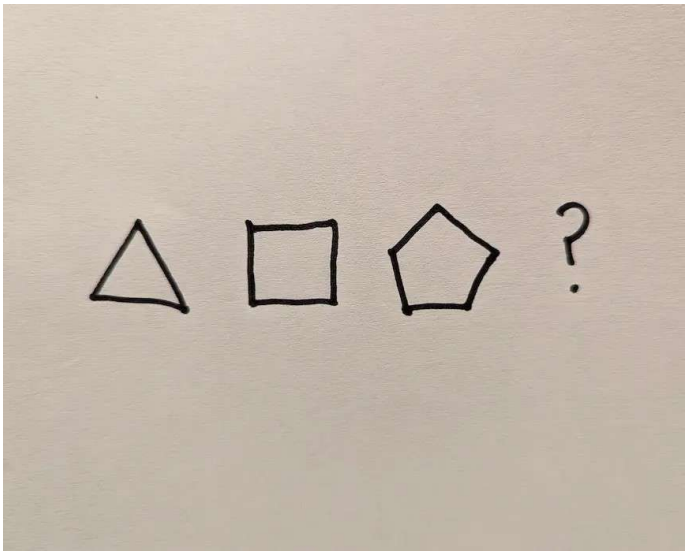

Figura 3.5: Imagen de entrada a Gemini en el ejemplo

'json' de Python permite la conversión entre diccionarios, cadenas JSON y archivos JSON. Para leer un archivo JSON en Python, se utiliza la función *json.load()* para cargar los datos del archivo en un diccionario. Para leer una cadena JSON en Python, se utiliza la función *json.loads()* para analizar la cadena y convertirla en un diccionario.

Para escribir datos en un archivo JSON, se utiliza la función *json.dump()* para escribir un diccionario en un archivo JSON. También se puede utilizar la función *json.dumps()* para convertir un diccionario en una cadena JSON y luego escribir esa cadena en un archivo. Ambas funciones permiten especificar la indentación para formatear el archivo JSON de manera legible. El módulo *json* proporciona una forma conveniente de manipular datos JSON en Python, facilitando el intercambio de datos entre aplicaciones y su almacenamiento en archivos.

3.3.2. RDF

Estas tripletas RDF se utilizan para almacenar información de manera estructurada y semántica, lo que permite una representación más rica y significativa de los datos en la web. A través de vocabularios y ontologías, como RDF Schema (RDFS) y Web Ontology Language (OWL), se establecen relaciones y significados precisos entre los términos utilizados en las tripletas RDF.

Las tripletas RDF son ampliamente utilizadas en la web semántica para diversas aplicaciones, como la descripción de recursos y metadatos en la web, la integración de datos de diferentes fuentes, la creación de motores de búsqueda más inteligentes y la construcción de sistemas de recomendación personalizados. Además, RDF proporciona un marco estándar y flexible para representar conocimiento y facilita la interoperabilidad entre diferentes sistemas y aplicaciones en la web.

El Framework de Descripción de Recursos (RDF, por sus siglas en inglés) es un

lenguaje de propósito general orientado a la representación de información en la web. Su uso se centra en el desarrollo de la web semántica, y tiene como finalidad describir los recursos de la misma de una manera no orientada a la legibilidad por parte de un humano, sino a la computación de la información contenido por un ordenador.

La web semántica es un proyecto de futuro en el que la información web tiene un significado exactamente definido y puede ser procesado por ordenadores. Por lo tanto, los ordenadores pueden integrar y usar la información disponible en la web. Más información sobre la web semántica puede ser encontrada en da Silva et al. (2007).

RDF está considerado como un lenguaje de metadatos, o "datos sobre datos", ya que con los símbolos de RDF añadimos metainformación a los datos que realmente nos interesan para poder interpretarlos de una manera exacta, es decir, de aquella que el autor de los datos (o, al menos, del autor del marcado RDF sobre estos datos) quería que estos fuesen interpretados.

RDF define los recursos mediante descripciones de los mismos, como puede verse en el listado 3.6. Puede encontrarse más información sobre la manera de la tecnología RDF de describir estos recursos en Champin (2002).

```
<rdf:Description
rdf:about="http://www.recshop.fake/cd/Empire Burlesque">
<cd:artist>Bob Dylan</cd:artist>
<cd:country>USA</cd:country>
<cd:company>Columbia</cd:company>
<cd:price>10.90</cd:price>
<cd:year>1985</cd:year>
</rdf:Description>
```

Figura 3.6: Ejemplo de un recurso RDF.

RDF no define clases de datos específicas para las aplicaciones. En vez de esto, el estándar RDF dispone de RDF Schema. Estos documentos definen nuevas clases, y relaciones entre ellas (herencia, agregación) de una manera muy similar a aquella con la que se acostumbra en la programación orientada a objetos.

En la Figura 3.7 podemos ver un ejemplo de un RDF Schema.

3.4. Desarrollo de la interfaz

PyQt, wxPython y Kivy son opciones populares para la implementación de interfaces gráficas, cada una con sus propias ventajas y desventajas.

PyQt es conocido por su completo conjunto de widgets, lo que te permite crear interfaces gráficas complejas y altamente personalizadas. Sin embargo, puede tener

```
<?xml version="1.0"?>
<rdf:RDF
xmlns:rdf= "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xml:base= "http://www.animals.fake/animals#">
<rdf:Description rdf:ID="animal">
<rdf:type
rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
</rdf:Description>
<rdf:Description rdf:ID="horse">
<rdf:type
rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
<rdfs:subClassOf rdf:resource="#animal"/>
</rdf:Description>
</rdf:RDF>
```

Figura 3.7: Ejemplo de un recurso RDF SCHEMA.

una curva de aprendizaje más pronunciada debido a su complejidad y sintaxis más compleja.

Por otro lado, wxPython ofrece una sintaxis más simple y fácil de entender, lo que puede ser beneficioso si estás empezando o prefieres un enfoque más directo. Aunque tiene menos widgets y funcionalidades avanzadas que PyQt, sigue siendo una opción sólida con una comunidad activa que proporciona soporte.

Kivy destaca por su diseño adaptable, diseñado para crear aplicaciones con interfaces gráficas que funcionan en una amplia gama de dispositivos. Utiliza un lenguaje de marcado declarativo que permite definir la interfaz de usuario de manera intuitiva y separada del código Python. Sin embargo, puede tener menos documentación y recursos disponibles en comparación con PyQt y wxPython.

3.5. Programación orientada a objetos

La Programación Orientada a Objetos (POO) ha ganado una popularidad significativa en la comunidad de programación debido a su capacidad para desarrollar aplicaciones más robustas, flexibles y fáciles de mantener. Este paradigma se basa en la organización de programas como una colección de objetos interconectados, cada uno con su propio conjunto de datos y funcionalidades. En este artículo, exploraremos los conceptos clave de la POO, cómo implementarla en diversos lenguajes y cómo aprovechar sus ventajas para construir aplicaciones sólidas y flexibles.

La POO es un paradigma de programación que se basa en la idea de clases y objetos. Se utiliza para estructurar programas de software en piezas simples y reutilizables de código, donde una clase actúa como una plantilla para crear múltiples instancias de objetos. Este enfoque invita a considerar las entidades dentro del con-

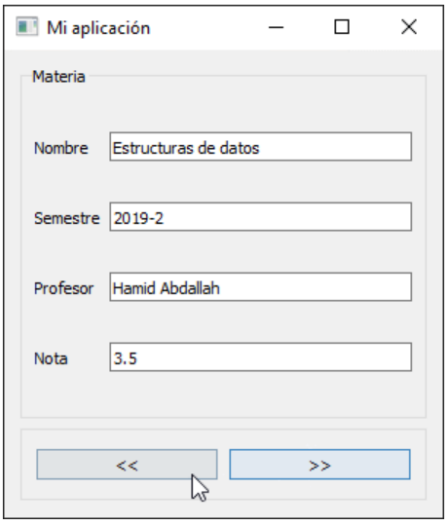

Figura 3.8: Ejemplo de interfaz generada con PyQT5

texto del problema a resolver, como libros, bibliotecarios y usuarios, y representarlas como objetos con propiedades y comportamientos.

La POO se inspira en la forma en que percibimos y entendemos el mundo que nos rodea. Cada entidad se convierte en un objeto con sus propios atributos y métodos, y la interacción entre estos objetos es fundamental. La encapsulación, abstracción, herencia y polimorfismo son los principios fundamentales de la POO que permiten crear aplicaciones más organizadas, reutilizables y mantenibles.

La POO permite la reutilización del código, evita la duplicación, protege la información a través de la encapsulación y facilita el trabajo en equipo al minimizar la posibilidad de duplicar funciones. Además, proporciona una estructura más clara y modular para el desarrollo de software, lo que facilita el mantenimiento y la escalabilidad a medida que los requisitos evolucionan.

La Programación Orientada a Objetos es esencial en el diseño de aplicaciones y programas informáticos modernos. Ofrece numerosas ventajas, como la reutilización del código, la modularidad y la facilidad de mantenimiento, lo que la convierte en una opción ideal para resolver desafíos de programación complejos. Sin embargo, requiere una planificación cuidadosa y un análisis detallado de los requisitos para aprovechar al máximo sus beneficios.

3.6. VPN

Una VPN (Red Privada Virtual) es una tecnología que establece una conexión segura y cifrada entre un dispositivo y una red privada remota a través de internet. Esto se logra mediante la creación de un túnel de comunicación virtual que encripta los datos transmitidos entre el dispositivo del usuario y el servidor VPN. Las VPN son utilizadas ampliamente por individuos y organizaciones para garantizar la privacidad, la seguridad y la accesibilidad de las comunicaciones en línea.

Las características y ventajas técnicas de las VPN incluyen:

Una VPN protege la privacidad al ocultar la dirección IP del usuario y encriptar los datos transmitidos. Esto previene la interceptación de datos por parte de terceros malintencionados, como hackers o agencias de vigilancia. La encriptación de extremo a extremo asegura que solo el dispositivo de origen y el servidor VPN puedan descifrar la información transmitida.

Las VPN utilizan diversos protocolos de seguridad, como IPsec (Protocolo de seguridad de Internet), SSL/TLS (Capa de sockets seguros/Protocolo de capa de transporte seguro), y OpenVPN, para establecer conexiones seguras y confiables. Estos protocolos garantizan la integridad de los datos y la autenticación de las partes involucradas en la comunicación.

Las VPN permiten a los empleados acceder de forma segura a recursos corporativos desde ubicaciones externas, utilizando conexiones cifradas. Esto es esencial para el trabajo remoto y las operaciones empresariales distribuidas, donde se necesita un acceso seguro a la red interna.

Las VPN son utilizadas por empresas para conectar múltiples ubicaciones geográficas en una red privada unificada a través de internet. Esto elimina la necesidad de redes privadas dedicadas y reduce los costos de infraestructura.

Al permitir que los usuarios simulen una ubicación diferente, las VPN facilitan el acceso a contenido y servicios en línea que podrían estar restringidos geográficamente. Esto es útil para usuarios individuales que desean acceder a sitios web o servicios que no están disponibles en su región.

Al utilizar VPN, las organizaciones pueden implementar estrategias de balanceo de carga y tolerancia a fallos para optimizar el rendimiento y la disponibilidad de sus servicios en línea.

Las VPN pueden integrarse con sistemas de filtrado de contenido y seguridad perimetral para proteger la red contra amenazas externas, como malware, phishing y ataques DDoS (denegación de servicio distribuido).

Capítulo 4

Desarrollo de prototipos

El capítulo actual tiene como objetivo presentar en orden temporal los distintos prototipos que se han desarrollado hasta llegar a la versión final que se presenta en el capítulo 5.

La estructura del capítulo muestra en orden lineal en el tiempo como se ha ido construyendo la arquitectura del sistema. De esta forma se puede ver como el modelo va pasando por distintos prototipos de forma incremental.

En este capítulo, también se pretende mostrar y hacer hincapié en los diferentes desafíos, contratiempos y retos a los que ha habido que hacer frente durante el desarrollo del prototipo, por qué se han originado y cómo se han solucionado.

Con todo ello, se pretende tener una visión global de lo que ha sido el trabajo de desarrollo del sistema. Partiendo de la situación que se muestra en el capítulo 2, y teniendo en cuenta el estudio teórico llevado a cabo en el capítulo 1 se trata de hacer un chatbot que facilite tanto como sea posible las tareas de extracción de la información en la terapia de reminiscencia. Para ello se usan las distintas herramientas detalladas en el capítulo 3 dando lugar al sistema que se presentará en el capítulo 5 [1].

El código asociado a los distintos prototipos se puede consultar en el repositorio de GitHub, y en concreto, en la carpeta de prototipos.

4.1. Planteamiento del problema

Como se expuso en el capítulo 1, el objetivo de este proyecto es construir una herramienta que facilite a los terapeutas la realización terapias de reminiscencia a personas con demencia. Se trata de un chatbot capaz de formular preguntas adecuadas, extraer información útil de ellas y generar nuevas preguntas para completar la información ausente.

[1] El repositorio con el código se puede consultar en github: https://github.com/NILGroup/TFG-2324-ChatbotCANTOR.

Con este objetivo en mente, se plantea un diseño iterativo que parta de un chatbot capaz de realizar preguntas predefinidas y sobre él ir iterando hasta conseguir un prototipo completo.

4.2. Prototipo usando preguntas predefinidas

La primera versión del prototipo tuvo como único objetivo ser una herramienta capaz de realizar una serie de preguntas predefinidas, y almacenar la información obtenida.

Para la selección de las mismas se optó por emplear la plantilla "Life History Templates" UK (2023). Estas preguntas desempeñan un papel crucial en el flujo de interacción del sistema, ya que son el esqueleto principal de la conversación con el usuario.

La plantilla UK (2023), concebida con el propósito de estructurar la recopilación de información sobre la historia de vida de un individuo, proporciona gran cantidad de aspectos relevantes. En primer lugar, al haber sido desarrollada por expertos en este tipo de tratamientos nos asegura que la información que se trata y que se extrae resulta útil a la hora de aplicar la terapia de reminiscencia. Por otro lado, la estructura en bloques con diferentes temáticas permite almacenar la información con mayor fiabilidad.

A partir de esta plantilla, se han seleccionado distintos campos tratando de asegurar que se aborden todos los aspectos clave, tales como eventos significativos, relaciones personales, intereses y preferencias, entre otros. Esto permite obtener una visión completa y detallada del individuo, lo cual es fundamental para proporcionar una atención personalizada y efectiva.

Una vez que se han identificado los campos pertinentes, se procede a la creación de un documento denominado "preguntas.txt", el cual sirve como guía durante la interacción con el paciente. En este documento se registran las preguntas específicas que se deben realizar en cada campo, así como la información deseada que se espera obtener a través de las respuestas del paciente. Esto facilita el seguimiento y la organización de la información recopilada, asegurando que se cubran todos los aspectos relevantes de la historia de vida del individuo.

Además, la plantilla "Life History Templates" incluye la recomendación de utilizar fotografías como herramienta para estimular la memoria y facilitar la obtención de información por parte del paciente. Esta estrategia, basada en la estimulación visual, ha demostrado ser efectiva para desencadenar recuerdos y promover la narración de experiencias pasadas.

Como parte de la implementación de nuestro prototipo, planeamos integrar esta funcionalidad en la sección dedicada a las imágenes (sección 4.5.6). Esto permitirá enriquecer la interacción con el paciente y mejorar la calidad de la información recopilada, contribuyendo así a una evaluación más completa y precisa de su historia de vida.

4.3. Prototipo utilizando Google BARD

El segundo hito en el desarrollo del chatbot fue mejorar el primer modelo haciendolo más inteligente y capaz de analizar las respuestas, identificar la información omitida y hacer preguntas especificas para obtener la información ausente o carente.

Tras el estudio realizado en la sección 3.2 se tomó la decisión de utilizar la API de BARD. La principal característica que llevó a tomar esta decisión es que se trata de una API ya entrenada capaz de crear texto de alta calidad similar al humano. Está hecha a medida para desarrolladores, permitiendo la automatización de la generación de contenidos para diversas necesidades, como en este caso la generación de texto para la implementación de un chatbot. Todo esto, supone una gran ventaja frente a otras herramientas como las distintas bibliotecas, cuyo uso implica desarrollar y generar nuestros propios algoritmos de generación del lenguaje, ó otras APIs que requieren un entrenamiento que da lugar a horas y horas de cómputo para obtener resultados, a lo sumo, tan buenos como los que ofrecía el uso de BARD sin necesidad de entrenamiento.

Para instalar BARD se utilizo el entorno de *Google Collaborate*. En primer lugar fue necesario instalar el paquete *bardapi* que se encarga de la interacción con Bard. Por otro lado, para poder utilizar la API de Bard hay que crear una variable de entorno, y utilizar una clave de autenticación. Una vez hechas las configuraciones pertinentes se puede usar la API como en el ejemplo que se muestra a continuación.

```
from bardapi import Bard

respuesta = Bard().get_answer("¿Cuáles son las mascotas más típicas
 en España?")

print(respuesta['content'])

>>> * Perros\n* Gatos\n* Peces\n* Pájaros (especialmente canarios y
 periquitos)\n* Conejos\n* Roedores (como hámsteres y cobayas)\n*
  Reptiles (como tortugas y serpientes)\n* Anfibios (como ranas y
   tritones)'
```

En la figura 4.1 se muestra un pequeño gráfico que pretende reflejar la arquitectura de este prototipo con el objetivo de que sea más fácil de entender su estructura

Figura 4.1: Arquitectura del prototipo con Bard

y funcionamiento.

Cabe recordar, que esta versión es un prototipo inicial muy lejos del objetivo final. Debido a los problemas que se explicaran más adelante nunca se llego a utilizar para las siguientes etapas, y en consecuencia nunca se llego a desarrollar una interfaz completa.

En la figura 4.1 vemos en el centro el módulo chatbot que es el que se encarga de gestionar el flujo de la conversación, interactuar con el usuario, llamar a Bard para analizar las respuestas y finalmente parsear y analizar la información.

Como se puede ver en la imagen el usuario se comunica con el chatbot mediante una interacción pregunta predefinida-respuesta. Para poder gestionar esta interacción con el usuario, se implementó una pequeña interfaz provisional desarrollada con widgets de IPyhton.

Estos widgets aparecen como cajas que solicitan información. Una vez rellanadas se envía la información al modelo para su análisis pulsando en el botón "Siguiente" que despliega un nuevo cuadro con la siguiente información solicitada.

Por ejemplo, en la imagen 4.2 podemos ver como el prototipo de chatbot solicita al usuario información como nombre, edad, ciudad, etc., y al final muestra un mensaje de despedida con los datos recopilados.

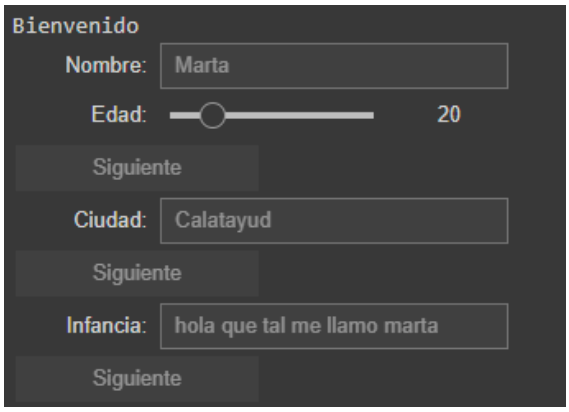

Figura 4.2: Muestra de la interacción con el usuario en el primer prototipo

Para el almacenamiento de la información se decidió usar *JSON* debido a su estructura legible y su amplia compatibilidad, suponen una elección tecnológica idónea para almacenar los datos de este sistema. Su capacidad para representar datos complejos de manera eficiente y su interoperabilidad garantizan una experiencia de desarrollo ágil y una gestión eficaz de la información de los usuarios. La API de Bard se usa para transformar las respuestas en un formato fácilmente convertible a un diccionario o un *JSON*. Para ello se usa el prompt que se muestra a continuación.

```
prompt=  "A partir del texto con información sobre una persona que
te voy a introducir a continuación, quiero que me devuelvas un texto
exactamente con el formato
Info[nombre_atributo:valor_atributo;nombre_atributo2:valor2]
```

```
Quiero que el texto este limpio, ningún tipo de formato especial
como negrita, ni cursiva. El texto es : "
```

Cabe destacar que BARD, pese a la solicitud de devolver el texto en un formato concreto, raramente satisface esa condición y usualmente añade mensajes de interacción con el usuario como *He escrito el texto exactamente en el formato que has pedido.* o *¿Qué puedo hacer por ti?*. Hay que tener en cuenta este aspecto a la hora de transformar el *string* en un json.

Para corregir esto se desarrolló una serie de funciones que parsearan el texto de forma manual, a partir de la respuesta dada por BARD. En concreto, la función $extraer\_info$ que extrae información de un texto en un formato específico, que es el pedido a BARD, y la devuelve como un diccionario de atributo-valor. Por otro lado, la función $parsear\_info$ procesa la información proporcionada por el usuario y actualiza el objeto Paciente.

Un ejemplo concreto de uso de Bard para este propósito es el siguiente.

```
texto = "Hola mi nombre es Marta, tengo 22 años y soy de Zaragoza."
respuesta = Bard.get_answer(prompt + texto)
print(respuesta['content'])
>> **Info[nombre:Marta;edad:22;ciudad:Zaragoza]**
¿Hay algo más que pueda hacer por ti?
```

Como se puede ver en el código anterior, BARD supone una herramienta útil para la extracción de información én base a las respuestas y la estructuración de la misma. A partir de la respuestas del modelo, y con las funciones que parsean la información que se han mencionado anteriormente, se crean las estructuras json que se encargan de manejar la información.

Todo este desarrollo se llevó a cabo durante los meses previos a diciembre de 2023. Fue ese mes cuando Google fortaleció la capacidad de Bard brindando habilidades más avanzadas de comprensión, razonamiento, resumen y codificación. De esta forma, paso a ser Gemini. Este cambio se oficializó en Febrero de 2024 y supuso la imposibilidad de usar BARD debido a las restricciones de uso según la localización de Gemini. Esta restricción limita el uso de la API de Gemini a una serie de regiones entre las que no se encuentra España ni ningún país de la Unión Europea.

En consecuencia, este prototipo pasó a un segundo plano y se centró el trabajo de los siguientes meses en encontrar qué API usar para el desarrollo del trabajo. Se barajaron distintas posibilidades.

Por un lado se estudió la API de GPT por sus buenas prestaciones y características, ya presentadas en el capítulo 3. Sin embargo, la API es de pago y para usarla en las versiones de bajo precio o gratuito es recomendable entrenarlo lo que suponía un coste en cómputo no asumible.

Por otro lado, se decidió estudiar distintos modelos de *Hugging Face* [2] entre los que destacó Gemma.

[2]https://huggingface.co/

4.4. Prototipo usando GEMMA

Al comenzar a desasrrollar este prototipo, se trató de seguir una estructura que permitiera reutilizar todo el código posible del prototipo anterior. Para ello, se trató de desarrollar una versión de nuevo en *GoogleCollaborate*. Posteriormente, se estudió la alternativa de la implementación de un modelo local. En el desarrollo de este prototipo se utilizaron diferentes bibliotecas de *HuggingFace*.

`AutoTokenizer` y `AutoModelForCausalLM` son clases proporcionadas por la biblioteca *HuggingFaceTransformers* que se utilizan para cargar y trabajar con modelos de lenguaje preentrenados.

1. `AutoTokenizer`: Esta clase se utiliza para cargar un tokenizador adecuado para el modelo de lenguaje preentrenado que se desea utilizar. El tokenizador se encarga de dividir el texto en unidades más pequeñas, como palabras o subpalabras, y convertirlas en vectores numéricos que el modelo de lenguaje puede entender. En nuestro contexto

   ```
   AutoTokenizer.from_pretrained(‘‘google/gemma-7b’’)
   ```

   se utiliza para cargar el tokenizador asociado al modelo de lenguaje $gemma-7b$ de Google.

2. `AutoModelForCausalLM`: Esta clase se utiliza para cargar el modelo de lenguaje preentrenado que se desea utilizar. El modelo de lenguaje es responsable de generar texto basado en las entradas proporcionadas. En nuestro contexto,

   ```
   AutoModelForCausalLM.from_pretrained(’’google/gemma-7b´´)
   ```

   se utiliza para cargar el modelo de lenguaje "gemma-7b"de Google, que es capaz de generar texto de manera condicional, es decir, generar texto secuencialmente teniendo en cuenta el contexto anterior.

En resumen, utilizaremos `AutoTokenizer` para cargar el tokenizador asociado al modelo de lenguaje preentrenado, y `AutoModelForCausalLM` para cargar el propio modelo. Estas clases son esenciales para preparar el modelo para su uso en la generación de texto o en otras tareas de procesamiento de lenguaje natural.

4.4.1. Google Collaborate

En primer lugar, para poder descargarnos el modelo se tiene que utilizar la biblioteca “os” para establecer las credenciales de autenticación de Hugging Face. Esto es necesario para acceder a modelos alojados en *HuggingFace* y para realizar operaciones como clonar repositorios de modelos.

A continuación es necesaria la carga de la clase *AutoTokenizer* y *AutoModelForCausalLM* de la biblioteca *Transformers* de Hugging Face para cargar un tokenizador y un modelo de lenguaje preentrenado. En este caso, el tokenizador y el modelo elegidos se cargan desde la dirección "$google/gemma-7b$".

Para poder realizar todas estas instalaciones se ha de configurar el entorno de ejecución de *GoogleCollaborate* de forma que se utilicen cuatro GPUs. Esto requiere seleccionar la versión "T4 GPU" del acelerador de hardware. Ejecutar el programa sin seleccionar esta opción impide la carga y descarga de los modelos y el programa y en consecuencia su uso.

Una vez realizada toda esta instalación, se puede utilizar el modelo de lenguaje cargado para generar texto a partir de una entrada específica. Sin embargo, en su versión gratuita *Collaborate* tiene limitado el uso de varios GPUs con lo que esta versión apenas pudo ser usada antes de que el entorno de ejecución limitara el uso de más de una GPU y en consecuencia, hubiera que descartar la alternativa de seguir desarrollando con $gemma-7$ en *Collaborate*.

En resumen, este código carga un modelo de lenguaje preentrenado, genera texto utilizando el modelo y realiza operaciones relacionadas con la gestión de modelos alojados en *HuggingFace*.

4.4.2. Linux

Las versiones locales de $gemma-2b$ y $gemma-7b$ en local han de ser instaladas en Linux debido a sus características. Requiere la instalación de las bibliotecas comentadas anteriormente así como de *HuggingFace* para poder acceder a los modelos.

En primer lugar, se optó por el modelo $gemma-2b$ pero debido a las limitaciones del mismo y a la alta exigencia de las tareas que se requieren en este proyecto, se tuvo que usar el modelo de 7 millones de parámetros. Con él, se llego a implementar una versión similar a la descrita en la sección 4.3 pero mediante el uso de $gemma-7b$ de *HuggingFace*.

Se amplió este modelo, tratando de alcanzar el siguiente reto: la generación de preguntas. Para ello, se desarrolló un módulo de generación de preguntas.

El módulo en concreto aborda la tarea de generar preguntas a partir de un texto dado utilizando herramientas de procesamiento de lenguaje natural. En concreto, se centra en la biblioteca *spaCy*, una poderosa herramienta para el procesamiento avanzado del lenguaje natural que ya ha sido presentada en la sección 3.1.2.

La generación de preguntas comienza con la tokenización y el análisis sintáctico del texto proporcionado. Para este propósito, se utiliza el modelo preentrenado de *spaCy* en español, denominado "$es\_core\_news\_sm$". Este modelo es capaz de analizar la estructura gramatical del texto y etiquetar cada token con información como el tipo de palabra (sustantivo, verbo, etc.) y la categoría gramatical.

Una vez que el texto ha sido analizado, el módulo procede a identificar los sustan-

tivos dentro del texto. Esto se logra mediante el análisis de las etiquetas gramaticales asignadas por *spaCy* a cada token. Cada sustantivo identificado se considera como un tema potencial para formular preguntas.

Con los sustantivos identificados, el módulo genera preguntas sobre la experiencia relacionada con cada uno de ellos. Por ejemplo, si el texto menciona la palabra "viaje", el módulo formulará la pregunta "¿Qué tal tu experiencia de viaje?".

Para el desarrollo de la interfaz de usuario, se consideraron dos enfoques diferentes: uno utilizando el framework Flask para crear un servidor web y otro utilizando la biblioteca Tkinter para construir una interfaz gráfica de usuario (GUI) de escritorio.

Con Flask, se optó por crear un servidor web que presenta una página web con un formulario. Flask maneja las rutas y funciones asociadas, permitiendo que los usuarios ingresen información en el formulario y envíen los datos al servidor. Esto resulta en una interfaz basada en la web que puede ser accesible a través de un navegador.

Por otro lado, con Tkinter, se creó una interfaz gráfica de usuario (GUI) de escritorio. Tkinter permite crear ventanas, etiquetas y botones para construir la interfaz. En este caso, se diseñó una ventana con una etiqueta de pregunta y dos botones de opción ("Sí" y "No"). Esta interfaz se ejecuta localmente en la máquina del usuario y es independiente del navegador web.

Este prototipo que supone un incremento respecto al anterior también tuvo que ser descartado sin llegar a desarrollar completamente algunas funcionalidades y módulos, como por ejemplo la interfaz o la generación de preguntas. Este parón se debió a un problema hardware. Al instalarme Linux, y debido a restricciones de espacio en el disco duro, tuve que utilizar una partición no demasiado grande. El gran tamaño de los modelos, en concreto del modelo $gemma - 7b$ ocupó gran parte del espacio de la partición. Pocas semanas después del comienzo del desarrollo de este prototipo tuvo que descartarse debido a la necesidad de ejecutarlo en Linux, acompañada de las limitaciones de espacio en mi ordenador que hicieron que no pudiera seguir desarrollandolo en ese sistema operativo.

4.5. Prototipo usando Gemini

A lo largo de los meses se estudiaron múltiples modelos y alternativas a BARD que no resultaban lo suficientemente eficaces. Por el contrario, otras sí resultaron eficaces pero pese a dar lugar a pequeños prototipos no pudieron seguir desarrollándose debido a los diversos problemas que se han expuesto a lo largo de este capítulo. Este estudio, implementación de modelos y desarrollo de prototipos fallidos conllevó horas de tiempo y esfuerzo. Derivó en meses centrados en análisis de alternativas en lugar de en el propio desarrollo del código en sí, por lo que finalmente hubo que decantarse por la API que resultaba más eficaz y fiable y que daba mejores resultados: la API de Gemini. Como ya se ha comentado antes, esta alternativa se descartó en un primer momento por no estar disponible en España. Sin embargo este problema se pudo solventar fácilmente mediante el uso de VPN.

Esta sección se centra en explicar como fue el desarrollo de este prototipo, y se deja para el capítulo 5 un estudio más profundo del resultado final así como los aspectos más técnicos.

Como ya se ha mencionado anteriormente, el segundo hito a conseguir es la construcción de una herramienta que sea capaz de analizar las respuestas, identificar la información omitida y hacer preguntas específicas para obtener la información faltante. Estas tres tareas se llevan a cabo usando la API de gemini y en concreto, el modelo $gemini - pro$, que es la versión de gemini encargada del procesamiento de texto.

La estructura de las respuestas de este modelo es la siguiente.

```
{
  "candidates": [
  {
    "content": {
      "parts": [
      {
        "text": string
      }
      ]
    },
    "finishReason": enum (FinishReason),
    "safetyRatings": [
    {
      "category": enum (HarmCategory),
      "probability": enum (HarmProbability),
      "blocked": boolean
    }
    ],
    "citationMetadata": {
      "citations": [
      {
        "startIndex": integer,
        "endIndex": integer,
        "uri": string,
        "title": string,
        "license": string,
        "publicationDate": {
          "year": integer,
          "month": integer,
          "day": integer
        }
      }
      ]
    }
  }
```

```
    ],
    "usageMetadata": {
        "promptTokenCount": integer,
        "candidatesTokenCount": integer,
        "totalTokenCount": integer
    }
}
```

Listing 4.1: Estructura de una respuesta de Gemini

- **text** El texto generado.
- **finishReason** El motivo por el que el modelo dejó de generar tokens. Si está vacío, el modelo no dejó de generar los tokens. El motivo puede ser cualquiera de los siguientes:
    1. *FINISH_REASON_UNSPECIFIED*: no se especifica el motivo de finalización.
    2. *FINISH_REASON_STOP*: punto de detención natural del modelo o secuencia de detención proporcionada.
    3. *FINISH_REASON_MAX_TOKENS*: se alcanzó la cantidad máxima de tokens especificada en la solicitud.
    4. *FINISH_REASON_SAFETY*: la generación del token se detuvo porque la respuesta se marcó por motivos de seguridad. Hay que tener en cuenta que Candidate.content está vacío si los filtros de contenido bloquean el resultado.
    5. *FINISH_REASON_RECITATION*: la generación del token se detuvo porque la respuesta se marcó para citas no autorizadas.
    6. *FINISH_REASON_OTHER*: todos los demás motivos que detuvieron el token
- **category** La categoría de seguridad para la que se configura un umbral. Los valores aceptables son los siguientes: Haz clic para expandir las categorías de seguridad
    1. *HARM_CATEGORY_SEXUALLY_EXPLICIT*
    2. *HARM_CATEGORY_HATE_SPEECH*
    3. *HARM_CATEGORY_HARASSMENT*
    4. *HARM_CATEGORY_DANGEROUS_CONTENT*
- **probability** Los niveles de probabilidad de daños en el contenido.
    1. *HARM_PROBABILITY_UNSPECIFIED*
    2. *NEGLIGIBLE*
    3. *LOW*

4. *MEDIUM*
5. *HIGH*

- **blocked** Una marca boolean asociada con un atributo de seguridad que indica si la entrada o salida del modelo se bloqueó. Si blocked es true, el campo errors en la respuesta contiene uno o más códigos de error. Si blocked es false, la respuesta no incluye el campo errors.
- **startIndex** Un número entero que especifica dónde comienza una cita en el contenido.
- **endIndex** Un número entero que especifica dónde termina una cita en content.
- **url** Es la URL de una fuente de cita. Los ejemplos de una fuente de URL pueden ser un sitio web de noticias o un repositorio de GitHub.
- **title** Es el título de una fuente de cita. Los ejemplos de títulos de origen pueden ser los de un artículo de noticias o un libro.
- **license** Es la licencia asociada con una cita.
- **publicationDate** La fecha en que se publicó una cita. Sus formatos válidos son YYYY, YYYY-MM y YYYY-MM-DD.
- **promptTokenCount** Cantidad de tokens en la solicitud.
- **candidatesTokenCount** Cantidad de tokens en las respuestas.
- **totalTokenCount** Cantidad de tokens en la solicitud y las respuestas.

Para el desarrollo del prototipo hay que tener en cuenta esta estructura. En concreto, resultan especialmente interesantes dos campos. Por un lado, el campo *candidates* que nos permite saber si se han generado respuestas correctamente. En distintos puntos dónde se usa la API de *gemini* hay que tener en cuenta que no necesariamente se obtiene siempre una respuesta para manejar la información de una forma u otra. Por otro lado, el campo *text* qué es el qué contiene la respuesta en sí.

```
response = model.generate_content("¿Qué es la terapia de
reminiscencia?")
print(response.text)
>>>'**Definición:**\n\nLa terapia de reminiscencia es un tipo de
psicoterapia que utiliza recuerdos personales como herramienta para
mejorar el bienestar cognitivo, emocional y social.\n\n**Objetivo:

*\n\nEl objetivo de la terapia de reminiscencia es ayudar a las
personas a recordar y dar sentido a sus experiencias pasadas,
promoviendo así:\n\n* Mejora del estado de ánimo\n* Reducción de
la ansiedad y la depresión\n* Apoyo a la autoestima y la identidad
```

```
\n* Conexión con el presente y el futuro\n\n**Proceso:**\n\nLa
terapia de reminiscencia implica:\n\n* **Sesiones guiadas:** Un
terapeuta guía al cliente a través de actividades que estimulan
los recuerdos, como contar historias, mirar álbumes de fotos o
escuchar música.\n* **Creación de una línea de vida:** El
terapeuta ayuda al cliente a crear una línea de vida visual
o escrita de su vida, destacando eventos importantes y hitos.
\n* **Discusión y reflexión:** El terapeuta y el cliente
discuten los recuerdos y cómo se relacionan con el presente
y el futuro.\n* **Exploración de temas:** La terapia se
centra en temas específicos, como la identidad, las
relaciones, los logros y las pérdidas.\n\n**Beneficios:
**\n\n* **Mejora de la memoria:** Estimula la función
cognitiva y mejora la memoria.\n* **Regulación emocional:**
Ayuda a las personas a procesar y regular sus emociones.\n*
**Reducción del aislamiento:** Fomenta la interacción social
y la sensación de pertenencia.\n* **Mejora de la autoestima:**
Ayuda a las personas a reconocer sus fortalezas y logros.\n*
**Prevención del deterioro cognitivo:** Puede retrasar o
prevenir el deterioro cognitivo en los adultos
mayores.\n\n**Aplicaciones:**\n\nLa terapia de reminiscencia
se utiliza comúnmente para:\n\n* Personas mayores con
demencia o deterioro cognitivo\n* Adultos que experimentan
soledad o aislamiento\n* Personas que luchan con la pérdida
o el trauma\n* Individuos con problemas de salud mental
como depresión o ansiedad'
```

Como se puede ver proporciona respuestas completas y bien redactadas.

4.5.1. Análisis de las respuestas e identificación de la información omitida

El proceso de análisis de respuestas implica evaluar las respuestas recibidas a las preguntas planteadas durante la conversación. En nuestro caso se lleva a cabo mediante el uso del modelo y una serie de funciones que reciben como entrada la respuesta del usuario y devuelven un *json* con la información asociada. Para ello, las preguntas predefinidas se almacenan en un fichero junto con una serie de campos que representan la información mínima que se quiere obtener de cada uno de ellos como se puede ver a continuación:

```
¿Tienes hijos? : [Nombre hijos,edad hijos,recuerdos con hijos]
```

El modelo, gracias a un prompt y a una llamada adecuada, se encarga de extraer la información y asociarla a cada campo, así como de indicar si hay información

faltante. Pese a que se pide que la salida se devuelva como un *json* esta información ha de ser parseada manualmente debido a las alucinaciones y otros fenómenos que pueden dar lugar a errores. Otro suceso que ha de ser tenido en cuenta es la posibilidad de no generación de respuesta por parte de gemini. Es por eso que hay que tener presentes en todo momento los distintos campos explicados en la sección 3.2.5, para tener un control de qué esta ocurriendo en todo momento.

```
pregunta = ¿Qué cosas te gusta hacer en tu tiempo libre?
respuesta = Tengo muchas aficiones. Por ejemplo me gusta mucho jugar
 al tenis. También disfruto de jugar al béisbol.
prompt = f"Generame un json con el formato campo:información
asociada a ese campo con la información que puedas obtener de
esta respuesta {pregunta.respuesta}"
campos = [aficiones]
>>>[{'text': '‘‘‘json\n{\n  "Aficiones": [\n    "Tenis",\n
 "Béisbol"\n  ]\n}\n‘‘‘'}]
```

Analizando el código anterior podemos estudiar cómo se comporta el modelo en la tarea de extracción de la información. A partir de la pregunta, la respuesta y el prompt dados, obtenemos una respuesta del modelo. Esta respuesta muestar como la de extracción de la información se ha realizado de forma correcta, sin embargo, el formato no es exactamente el pedido. Por otro lado, el comportamiento de *gemini* no es determinante, es decir, no siempre da las mismas respuestas y en ocasiones (aunque no son las mayoritarias) ocurren sucesos como que no se generan respuestas, las respuestas no se adecuan a la situación o se producen alucinaciones (fenómeno explicado en la sección 2.5).

Para solventar este problema, el código anterior se distribuye entre distintas funciones que se encargan de checkear el correcto funcionamiento del programa, y una vez obtenida una respuesta adecuada parsear la misma eliminado caracteres no deseados como espacios, comillas o saltos de línea. De esta forma, siguiendo el ejemplo anterior, la respuesta de *gemini* sería transformada hasta obtenerse como resultado un diccionario campos actualizado como el siguiente.

```
campos = {'aficiones': ['Tenis', 'Béisbol']}
```

Para identificar la información ausente se actualiza el prompt que se había usado anteriormente como sigue.

```
prompt = f"Tras preguntar {pregunta} en relacion con {clave} me han respondido
 {respuesta}. "
prompt = prompt + f"Dada esa información, quiero que devuelvas cualquier
 información válida sobre {clave} y en caso de que no se haya podido obtener
  ninguna información válida, devolver No Encontrado"
```

De esta forma, podemos saber qué información no ha sido encontrada en el texto para poder indagar y generar nuevas preguntas. A continuación se muestra un ejemplo.

```
>>>¿Tienes hijos?
>>> Sí tengo hijos pero no estoy seguro de dónde están ahora.
Mi hija Lucía es médico y el mayor es Juan.
>>> {'Nombre hijos': ['Lucía','Juan'], 'edad hijos': 'No
Encontrado', 'recuerdos con hijos' : 'No Encontrado'}
```

Este prototipo también añade información adicional que haya podido obtener de la respuesta y que no se pueda asociar a los campos predefinidos. Para ello, utiliza el prompt original que no pasaba las claves, y al que se pedía que extrajera la información posible. Si alguna de las claves del diccionario de información extra no aparece ya en el diccionario se añade.

4.5.2. Generación de preguntas para obtener la información ausente

La generación automatizada de preguntas es un componente clave del sistema diseñado. Este proceso implica la creación de nuevas preguntas dirigidas a recopilar información que no ha podido ser obtenida en las preguntas predefinidas. El proceso de generación de preguntas comienza tras la identificación de un campo al que no se ha podido asociar ningún tipo de valor o sobre el que no se ha obtenido información. En ese momento y gracias a *gemini* se generan una serie de preguntas asociadas asociadas a los campos que no han sido rellenados tales como las que se muestran a continuación.

```
>>>'¿Cuáles son sus edades?'
>>> '¿Cómo describe la dinámica de su familia?'
>>> '¿Qué recuerdos tiene con sus hijos?'
```

Estas preguntas se van lanzando al usuario hasta rellenar los campos faltantes o hasta agotar el limite de preguntas asociadas a un mismo tema que se va a hacer al usuario.

Una vez el usuario responde a una pregunta, se extrae la información de la misma y se parsea hasta rellenar el diccionario con los campos, indicando los campos predefinidos para los que no se ha encontrado información. El siguiente paso en el flujo del programa es pasar toda esta información (pregunta, respuesta y información extraida) al módulo de generación de preguntas. Este módulo partirá de aquellos campos para los que no se ha encontrado información y generará una serie de preguntas que poder lanzar al usuario. La generación de preguntas se hace mediante la función *generate_question*(). En concreto, se muestra el siguiente ejemplo.

```
>>>['Cómo has definido el concepto de amistad a lo largo de tu vida',

'Qué cualidades valoras más en los amigos',
```

```
'Cuáles son algunos de los momentos más significativos que has
compartido con amigos',

'Cómo han dado forma las amistades a la persona que eres hoy',

'Cómo mantienes tus amistades',

'Cómo equilibras el tiempo con amigos y otras responsabilidades',

'Has tenido alguna dificultad o malentendido en tus amistades',

'Cómo los has abordado',

'Cómo ha evolucionado tu círculo de amigos a lo largo del tiempo',

'Hay amistades que has perdido y otras que has ganado',

'Crees que es importante tener amistades diversas y por qué',

'Cómo han influido las redes sociales en tus amistades',

'Crees que han mejorado o perjudicado las conexiones',

'Tienes límites o pautas para las interacciones en las redes
 sociales con tus amigos',

'Alguna vez has perdido una amistad debido a una interacción
en las redes sociales',

'Cuáles son tus expectativas y necesidades en tus amistades',

'Te sientes apoyado y comprendido por tus amigos',

'Hay alguna área en la que te gustaría ver más apoyo',

'Alguna vez has tenido que poner fin a una amistad',

'Cuáles fueron las razones',

'Crees que las amistades son esenciales para una vida plena y feliz']
```

La función se encarga de hacer al modelo la solicitud de generación de preguntas hasta que se obtenga una respuesta válida (es decir, si no existen candidatos de respuesta se vuelve a llamar al modelo). Una vez generada la respuesta por parte

del módulo se trasforma en una lista de preguntas con el formato correcto. Estas preguntas se lanzaran al usuario y pasaran por las funciones de extracción de la información explicadas en la sección anterior, y sólo se seguirán haciendo mientras el campo este incompleto. Por otro lado, se lleva un control acerca de qué campos se han hecho ya preguntas adicionales para, en caso de que al llegar al final de la lista sin haber obtenido información, no se vuelva a generar otra lista.

4.5.3. Interfaz e interacción con el usuario

El siguiente hito en el desarrollo del chatbot es desarrollar la interfaz que permita al usuario interactuar de una forma que resulte sencilla y práctica. Paralelamente a todo el trabajo descrito anteriormente, y durante el estudio de las diferentes alternativas de interfaces, APIs y modelos se estudió la posibilidad de desarrollar la interfaz con Telegram.

Esta opción fue encontrada al buscar alternativa a BARD o Gemma, y hallarse diversos chatbots implementados con Rasa y que adaptaban facilmente su interfaz a Telegram. Aunque la capacidad de Rasa de generación de texto no cumplía los requisitos requeridos para ser considerada en este proyecto, se encontró especialmente interesante la fácil integración de chatbots con la interfaz de Telegram.

Como se puede ver en la figura 4.3 la interfaz generada de esta forma presenta númerosas ventajas. En primer lugar, Telegram es una plataforma de mensajería ampliamente utilizada en todo el mundo, lo que garantiza un gran alcance y accesibilidad para los usuarios y reduce la curva de aprendizaje. Además, los usuarios no sólo pueden acceder fácilmente desde su escritorio, si no también desde sus dispositivos móviles, lo que simplifica su adopción y uso.

Esta forma de generar la interfaz implica el desarrollo de manejadores que controlen el funcionamiento según el tipo de mensaje. Se utiliza la API de Telegram a través del módulo "telebot" para interactuar con el servicio de mensajería y gestionar los mensajes entrantes y salientes. Además, se inicia un bucle de escucha para que el bot esté constantemente activo y pueda responder a los mensajes de los usuarios en tiempo real.

Telegram también ofrece una variedad de funcionalidades integradas que pueden ser útiles para un chatbot de terapia de reminiscencia, como el envío de fotos de forma sencilla. Esto permite que la experiencia del usuario sea más rica y variada, siendo mucho más fiel a la plantilla de generación de historias de vida mencionada en la sección 4.2. En consecuencia, puede mejorar la efectividad del chatbot en la prestación de servicios de apoyo emocional.

4.5.4. Telegram

Para conectar Telegram con el chatbot desarrollado a través de la API de *gemini*, fue necesario obtener un token de Telegram. Es importante mencionar que este paso forma parte de la configuración inicial y no requiere intervención por parte del

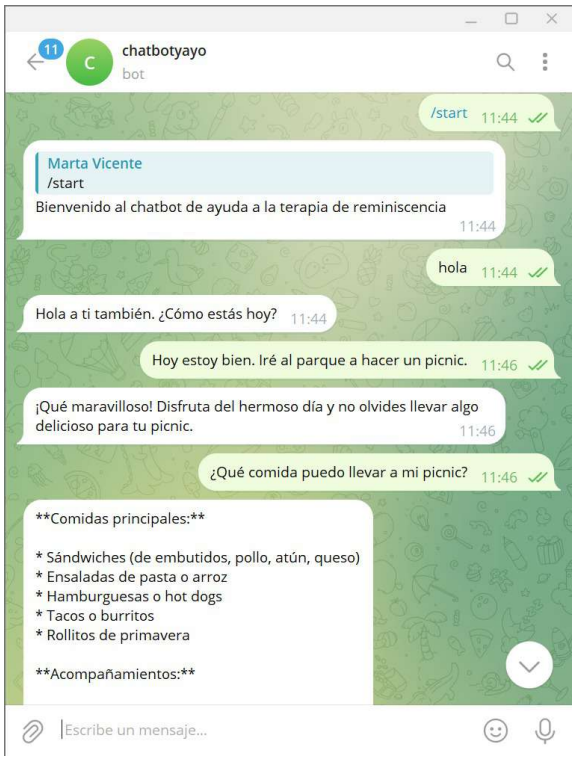

Figura 4.3: Ejemplo de uso de BARD.

usuario final durante la puesta en marcha.

Este token se obtuvo a través del usuario de telegram *@BotFather* y con el comando *\newbot* y *\token*. Una vez con el token se creo el *chatbotyayo* con nombre de usuario *@mavice07_bot*. Para comenzar a interactuar con *chatbotyayo* basta con buscar ese usuario en Telegram y enviar el comando *\start*.

Una vez obtenido el token y finalizadas todas las configuraciones se implementa la lógica del sistema. Se definen tres manejadores de mensajes.

- El manejador *cmd_start*() para el comando start.
- El manejador *bot_mensajes_text*() se activa cuando un usuario envía un mensaje de texto al bot.
- El manejador *photo*() se activa cuando un usuario envía una foto al bot.

Finalmente, se inicia el bucle de escucha del bot *bot.infinity_polling*() para que esté constantemente esperando y respondiendo a los mensajes de los usuarios. El flujo de la información a través de los manejadores se explicará con más detalle en el capítulo 5 que muestra la implementación final.

4.5.5. Clase Pregunta

La decisión de desarrollar la interfaz del prototipo integrando Telegram y *gemini* para tener una interfaz en Telegram tuvo como consecuencia la necesidad de crear una clase "Pregunta". Se crea para almacenar de manera integral toda la información asociada con una pregunta en un programa, especialmente en el contexto de la interacción con Telegram mediante textos. En versiones anteriores del desarrollo, cada vez que se requería información adicional, se realizaban preguntas y el flujo no avanzaba hasta completar estas preguntas extra o obtener la información necesaria. Al integrar esta funcionalidad con Telegram, se necesita devolver desde la función principal la información para que la interfaz pueda mostrar la pregunta, lo cual puede interrumpir el flujo original del código. Esta integración conlleva a una refactorización que dio lugar a la creación de la clase "Pregunta".

La clase "Pregunta" encapsula una pregunta con su enunciado, respuesta, campos asociados y preguntas adicionales relacionadas. Su objetivo principal es proporcionar una estructura organizada para manejar la información de una pregunta específica y sus detalles. Permite actualizar la respuesta principal, indicar el estado de los campos (respondidos o no respondidos), agregar preguntas adicionales relacionadas y gestionar las respuestas a estas preguntas secundarias.

Además, la clase "Pregunta" ofrece métodos para actualizar la respuesta, marcar campos como respondidos, agregar preguntas adicionales, gestionar las respuestas a estas preguntas secundarias y obtener una representación textual de la instancia de la clase.

La representación de una pregunta en un momento del flujo de una conversación sería la siguiente.

```
Enunciado: ¿Qué rutinas diarias son importantes para ti?

Respuesta: Tengo muchas aficiones. Por ejemplo me gusta mucho jugar al
tenis. También disfruto de jugar al béisbol.

Campos: :
{'Aseo personal': 'ducharse a diario' , 'Limpieza del hogar':
'No Encontrado' , 'cuidado de la salud' : 'ir al médico rutinariamente'}

Preguntas Extra:
['Con qué frecuencia se ducha',
'Cuánto suele durar su ducha',
'Qué productos utiliza para ducharse (champú, acondicionador, jabón)',
'Tiene algún tipo de piel específico (grasa, seca, mixta)',
'Qué productos utiliza para limpiar su piel',
'Utiliza algún producto de hidratación o protección solar',
'Qué tipo de cabello tiene (liso, rizado, teñido)',
'Con qué frecuencia se lava el cabello',
'Qué productos utiliza para lavarse y acondicionarse el cabello',
```

```
'Con qué frecuencia se corta y lima las uñas',
'Utiliza algún esmalte o producto para las uñas',
'Ha notado algún problema con las uñas (por ejemplo, fragilidad,
 decoloración)',
'Cuántas veces al día se cepilla los dientes',
'Utiliza hilo dental o enjuague bucal',
'Ha notado algún problema dental (por ejemplo, caries,
enfermedad de las encías)',
'Utiliza desodorante o antitranspirante',
'Tiene alguna rutina de exfoliación o depilación',
'Utiliza lociones corporales o perfumes',
'Cuánto suele dormir',
'Tiene alguna dificultad para dormir',
'Cómo afecta su falta de sueño a su rutina de aseo personal']

Extra: {'Aseo personal': True , 'Limpieza del hogar': False ,
'cuidado de la salud' : False}

Respuestas Extra: {}
```

Un objeto pregunta por un lado, almacena la pregunta principal que es la que viene predefinida junto con la respuesta y los campos. Si una vez realizada esa pregunta y analizada la respuesta quedan campos sin responder, se generan preguntas extra asociadas a ese campo, que se almacenan en el atributo "Preguntas Extra". Las preguntas extra que se van haciendo se almacenan en "Respuestas extra" con el formato *Pregunta* : *respuesta*. Todas las respuestas dadas se analizan y si se encontrara un valor para los campos que faltan, se eliminarían el resto de preguntas extra y se pasaría a ver el siguiente campo. Llevar un control acerca de qué campos se han generado ya preguntas extra nos permite no volver a indagar sobre un tema del que ya se ha conversado en profundidad.

Cuando se estudian los campos de un objeto pregunta para comprobar que se ha obtenido toda la información que se necesitaba y poder pasar a la siguiente, sí hay un campo para el que no se ha encontrado valor, pero para el que sí se han generado preguntas extra y preguntado sobre ellas, simplemente se da la información por no encontrable y se continua con el flujo de la conversación. Además, esto nos permite mantener un mejor flujo de la conversación, ya que todas las preguntas relacionadas con un mismo tema se harán en un orden lógico y no dando saltos entorno a los diferentes temas a lo largo de la conversación.

4.5.6. Extracción de información a partir de imagenes

La API de Gemini tiene muchas funcionalidades que permiten del chatbot una herramienta completa. En concreto, gracias a la interfaz elegida y al modelo *gemini-pro-vision* se ha implementado una funcionalidad extra respecto a las que estaban planteadas originalmente. Esta funcionalidad es la extracción de información a partir

de imagenes. En cualquier momento, el usuario puede adjuntar una foto. El programa se encarga de analizar la fotografía y a partir de ella hacer preguntas que permitan extraer más información que la obtenida únicamente con las preguntas.

Mediante el reconocimiento y clasificación de objetos, personas y lugares en imágenes, el procesamiento de imagenes puede identificar elementos significativos que pueden evocar recuerdos en los pacientes.

El análisis de imágenes también puede proporcionar contexto y significado a las experiencias pasadas. Examinar el contenido visual de las fotografías, puede ayudar a contextualizar recuerdos específicos, facilitando así la conversación y la narración de historias durante las sesiones de terapia. La selección cuidadosa de imágenes personalizadas para cada paciente puede estimular recuerdos y promover interacciones significativas.

Figura 4.4: Imagen de entrada para el análisis de la misma por *gemini-pro-vision*

A partir de la imagen 4.4 se puede ver cuál es la respuesta de *gemini*. La API es capaz de identificar el contexto y las personas en la imagen y generar preguntas especificas sobre ellas. Las imágenes se pueden adjuntar en cualquier momento junto con las respuestas. El prompt que se pasa al modelo en este caso indica que se describa la imagen y se genere una pregunta específica de forma que, si el usuario no había sido capaz de responder a una pregunta, apoyándose de la foto sea capaz de proporcionar algo de información aunque sea más especifica.

Es importante mencionar, que el comportamiento de Gemini no es siempre el mismo de forma que para la misma entrada se pueden obtener diferentes salidas. De hecho para la imagen 4.4 en distintas ejecuciones se obtienen entre otras estas salidas.

```
La imagen muestra a una familia de tres generaciones. Hay abuelos,
padres y nietos. Todos estan sonrientes y pareen estar disfrutando de
la compañía de los demás. Están sentados alrededor de una mesa en el
jardín, y hay comida en la mesa. Parece que estan celebrando algo.
```

```
¿Qué estan celebrando?

La imagen muestra a una familia de 10 personas. Hay 4 generaciones
presentes: los abuelos, los padres, los hijos y los nietos. Todos están
sonrientes y parecen estar disfrutando de la comida al aire libre.

¿Cuál es el motivo de la reunión familiar?
```

La información obtenida a partir de las respuestas que diera el usuario a las preguntas que aparecen al final de la descripción son las que se almacenarían y analizarían siguiendo el mismo flujo que las respuestas asociadas a preguntas normales.

Capítulo 5

Diseño de un chatbot de ayuda a la terapia de reminiscencia

Este capítulo tiene como objetivo presentar la versión final desarrollada teniendo en cuenta la problemática detallada en el capítulo 1.2. Para ello, se describirá en profundidad cada uno de los componentes básicos de la arquitectura del sistema, y se presentara esta en sí misma. Por otro lado, se explicará tanto el proceso de puesta en marcha como las herramientas necesarias para ese mismo objetivo.

Con la finalidad de permitir el estudio de los componentes de forma práctica, es decir, usando el prototipo, este capítulo comenzará por la explicación de la puesta en marcha del mismo. A continuación, se dará una idea global de la arquitectura del sistema y finalmente se irá explorando en cada sección, cada uno de los módulos que la componen.

5.1. Herramientas y puesta en marcha

Para la puesta en marcha el primer paso es obtener la *API Key* para lo que es necesario el uso de una VPN.

5.1.1. VPN

Debido a las restricciones geográficas actuales de la API de Gemini para poder usarla es necesario el uso de una VPN. En concreto y para el desarrollo de este proyecto la conexión a la VPN se ha realizado mediante la herramienta *hide.me VPN*. Esta herramienta crea un túnel seguro utilizando protocolos VPN, oculta nuestra IP real con una suya y cifra todo el tráfico de internet que pasa por este túnel para que podamos navegar libremente. Además, *hide.me* está certificada como una VPN cero registros. Esto significa que no se almacena información de ningún tipo. El uso de la VPN es necesario tanto para obtener la *API Key* como para el uso de la misma.

Los países en los que se encuentra disponible *gemini* se pueden consultar en la web de la api de *gemini*.

5.1.2. Instalación de la API de *gemini*

Para comenzar a utilizar la API de *gemini* con Python, es necesario seguir estos pasos para instalar el SDK y configurar tu clave de API.

En primer lugar, instalamos el SDK [1] (Software Development Kit). La API de *gemini* está contenida en el paquete `google-generativeai` [2] en PyPI, por lo que el primer paso sera instalar esa dependencia.

```
!pip install -U google-generativeai
```

Para utilizar la API de *gemini*, se necesita una clave de API obtenida del Google AI Studio. Una vez que se tenga la clave, puede configurarse para que el SDK la utilice:

```
import google.generativeai as genai
from google.colab import userdata

GOOGLE_API_KEY = userdata.get('GOOGLE_API_KEY')
genai.configure(api_key=GOOGLE_API_KEY)
```

Siguiendo con la arquitectura del sistema bastaría con modificar en el archivo *config.py* el valor de la *API KEY*.

5.1.3. Puesta en marcha

Una vez hechas todas las configuraciones necesarias explicadas en las secciones anteriores, para poder usar el chatbot sería necesario poner en ejecución el módulo *mibot.py* y enviar el comando *\start* al usuario *@mavice07_bot*. Es importante recordar, que para que el análisis de la información y el flujo de la conversación se desarrollen correctamente, hay que estar conectado a una VPN de una de las localizaciones en las que se encuentra disponible la API de *gemini*, y que como ya se ha comentado con anterioridad, se pueden consultar en la web de *gemini*.

5.2. Arquitectura del sistema

Con todo lo implementado, el sistema se puede representar como se ve en la figura 5.1.

[1] `https://ai.google/discover/generativeai/`
[2] `https://ai.google.dev/gemini-api/docs/available-regions?hl=es-419`

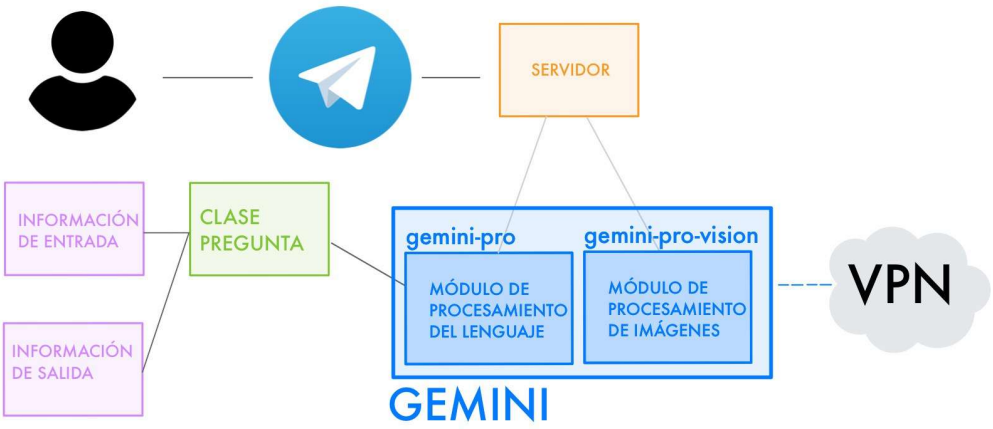

Figura 5.1: Arquitectura del sistema

5.2.1. Servidor

El módulo servidor se encarga de manejar el flujo de los mensajes del chatbot. Recibe los mensajes entrantes desde la plataforma de Telegram y coordina su procesamiento posterior. Este módulo actúa como el punto de entrada centralizado del sistema, y ha de mantenerse en ejecución durante todo el tiempo en el que se quiera usar el chatbot.

La lógica del módulo para manejar los mensajes sigue lo explicado en la sección 4.5.4.

Esta capacidad de coordinación del módulo servidor asegura una distribución eficiente de los mensajes entrantes optimizando así la velocidad de respuesta y la efectividad de las interacciones con los usuarios. Este manejo eficiente se hace a través del uso de tres manejadores, cada uno encargado de gestionar los tres tipos de mensajes de entrada: comandos, imagenes y texto.

- El manejador $cmd\_start()$ se activa únicamente cuando un usuario envía el comando $\backslash start$ al bot. Este manejador inicia la lógica del bot al cargar las preguntas utilizando la función $cargar\_preguntas()$ del módulo tfg y envía un mensaje de bienvenida al usuario.
- El manejador $bot\_mensajes\_text()$ se activa cuando un usuario envía un mensaje de texto al bot. Si el mensaje comienza con "\", se envía un mensaje de error indicando que el comando no está disponible. Si es un mensaje de texto estándar, se llama a la función $siguientePregunta()$ del módulo tfg y se analiza el mensaje como respuesta al último mensaje enviado por el bot. De esta forma se procesa la respuesta del usuario y se genera una respuesta apropiada.

- El manejador *photo*() se activa cuando un usuario envía una foto al bot. El manejador descarga la foto y la guarda en el sistema de archivos. A continuación, llama a la función *analizador_imagenes*() del módulo *imagenes* para analizar la imagen y generar una respuesta apropiada. El modelo de *gemini-pro-vision* carga la foto descargada y la analiza con el prompt que le indica que describa la imagen y genere una pregunta.

El módulo servidor sirve como el núcleo activo del chatbot, recibiendo mensajes desde Telegram y dirigiéndolos hacia los componentes adecuados del sistema para su procesamiento y respuesta. Su función es fundamental para mantener el flujo de comunicación entre los usuarios y el chatbot, facilitando una experiencia de usuario fluida y receptiva.

5.2.2. Módulo de procesamiento del lenguaje

El módulo del procesamiento de la información es el que usa *gemini*, y que por tanto requiere estar conectado a una VPN. En función del tipo de mensaje de entrada, imagen o texto, el servidor enviará la información al módulo de procesamiento del lenguaje o al módulo del procesamiento de imagenes.

Cuando se ha iniciado el chatbot y se han cargado las preguntas, el resto de mensajes de texto serán analizados por este módulo. La tarea principal del módulo es ir pasando una por una por todas las preguntas predefinidas rellenando la información y almacenando las respuestas.

Cuando el módulo hace una pregunta, almacena la respuesta y la manda a analizar con los prompts y funciones ya explicadas en la sección 4.5. Estas funciones se encargan de coger la respuesta del usuario , extraer la información útil y asociarla a campos predefinidos (además de añadir nuevos campos si hay información extra).

El manejo de la información se lleva a cabo a través de ficheros. Cuando el servidor recibe el comando *start* carga y crea las preguntas del fichero *preguntas.txt*. Una vez llevada a cabo toda la conversación con el usuario, la información obtenida y la información generada se guardan en el fichero de salida *información.txt*

En el módulo de procesamiento de texto se agrupa la mayoría de la funcionalidad del chatbot. Para entender mejor cuál es el flujo de la información vamos a ver las diferentes tareas que realiza a través de un ejemplo concreto.

- Control de la lógica del sistema y flujo de las preguntas.

  Cuando empieza la conversación el módulo se encarga de cargar toda la información a partir del fichero *preguntas.txt* y transformarla en objetos de la clase "Pregunta". A continuación, se encarga de manejar el flujo de la conversación a través de las preguntas predefinidas que acaba de cargar.

  Supongamos que nos encontramos en plena conversación y que se ha lanzado al usuario la pregunta "¿Cuáles son tu comida y bebida favoritas?" con campos

asociados ['comida favorita', 'bebida favorita']. Hemos obtenido la respuesta "Mi comida favorita son las lentejas pero detesto los garbanzos".

- Análisis de las respuestas.

  Una vez recibida la respuesta esta pasaría a ser analizada. Si llamamos al modelo usando los prompts indicados en 4.5.1 obtenemos la siguiente respuesta.

  ```
  ‘‘‘json\n{\n  "comida favorita": [\n    "Lentejas"\n  ],\n  "bebida
   favorita": [\n    "No Encontrado"\n  ], \n "comida detestada":[\n
    garbanzos]\n}\n‘‘‘
  ```

  La información se extrae de forma adecuada, e incluso se añade información adicional que no se corresponde con ningún campo predefinido. Esto ocurre gracias a una llamada adicional al modelo en la que se pide que añada información extra en caso de haberla encontrado, como se explica en 4.5.1

- Generación de los *json* con la información.

  El siguiente paso es enviar este valor a diferentes funciones que se encargan en transformar ese *string* en el suguiente *json*, que se almacena en el atributo *campos* del objeto "Pregunta" asociado.

  ```
  {'comida favorita': ['lentejas'], 'bebida favorita': ['No Encontrado'],
   'comida detestada': ['garbanzos']}
  ```

- Identificación de la información ausente.

  Cuando termina la fase de extracción de la información y se ha generado el *json* resulta sencillo comprobar para que campos no se ha obtenido información. Antes de pasar a la siguiente pregunta predefinida, se itera sobre todos los campos comprobando que no hay información ausente. En caso de encontrar un campo para el que no se ha encontrado valor. Se pasa a la etapa de generación de preguntas.

- Generación de preguntas adicionales.

  Como se puede ver, en nuestro caso no se ha obtenido información para el campo "bebida favorita" con lo que se generan preguntas para este campo como se explica en la sección 4.5.2. Con esto se actualiza el atributo "Preguntas extra" del objeto "Pregunta" y se pasa a formular estas preguntas una a una hasta que se rellene el campo "bebida favorita" o hasta que se acaben las preguntas extra generadas. En este segundo caso, no se generaran más preguntas si no que se continuará con el flujo normal de la conversación.

  Igual que ocurría con la extracción de información, la respuesta que da *gemini* necesita ser parseada para obtener las respuestas con un formato "limpio".

  Un ejemplo de generación de preguntas para el campo "bebida favorita". Supongamos que en la lista de preguntas generadas se encuentra la siguiente, y que al hacérsela al usuario se obtiene la respuesta mostrada a continuación.

```
¿Cuál es tu bebida favorita?
>>> Mi bebida favorita es el café, pero de jovén me gustaba la
 cerveza.
```

Esta pregunta y respuesta se almacenarían en el diccionario "respuestasExtra" y se analizarían según lo explicado anteriormente como una respuesta normal. Tras todo el proceso de extracción de la información el atributo campos queda como sigue.

```
{'comida favorita': ['lentejas'], 'bebida favorita': ['café'],
'comida detestada': ['garbanzos'], 'bebida juventud' : ['cerveza'] }
```

- Generación de feedback a las respuestas del usuario.

  Para tener una sensación de conversación más real y que el chatbot no se limite sólo a hacer preguntas, se le pide al modelo que genere cierto feedback para las respuestas. Por ejemplo, como se muestra en la figura 5.2.

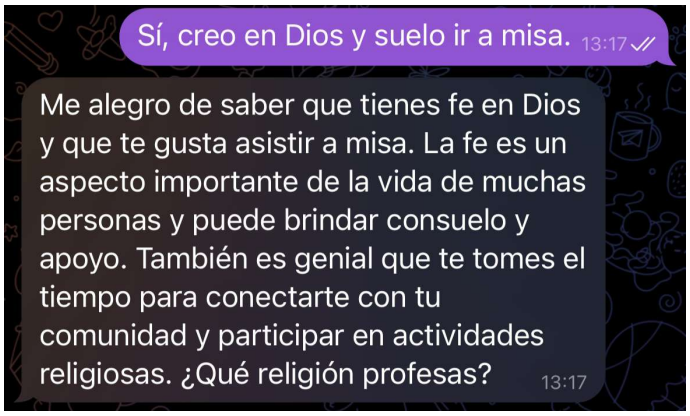

Figura 5.2: Ejemplo de generación de feedback a las respuestas

En cualquier momento, una respuesta puede venir acompañada de una imagen y en ese caso, se pasaría al módulo de procesamiento de imagenes.

Recapitulando, el modelo *gemini-pro* se utiliza para las siguientes tareas:

- Análisis de respuestas y transformación de la información extraída a cierto formato que se pueda parsear fácilmente.
- Generación de preguntas.
- Generación de feedback a las respuestas.

Adicionalmente, también se usará para la generación de historias de vida.

5.2.2.1. Generación de historias de vida

Finalmente, se añade la funcionalidad adicional de generar historias de vida. Lo idóneo sería generar las historias de vida con otros modelos como el mencionado en 2.7.1.1. Sin embargo, a modo de feedback final de la conversación se propone una historia que podría ser un prototipo de historia de vida. Esta historia es generada por el modelo pasandole un prompt tan simple como el siguiente.

```
prompt = f"Genera una historia sobre una persona con estos datos
 {datos}"
```

La entrada datos, en este caso es un *json* que concatena por cada una de las preguntas los *json* con información que han obtenido. Un ejemplo de generación de historia de vida con este método es la siguiente.

```
datos = {
"Nombre": "María",
"Apodo": "Marucha",
"Fecha de nacimiento": "15 de febrero de 1930",
"Lugar de nacimiento": "Barcelona, España",
"Padres": "Joaquín y Carmen",
"Hermanos": "Pedro y Ana",
"Anécdota de infancia": "Una vez me perdí en el parque y un policía
me ayudó a encontrar a mis padres.",
"Mejor amigo de la infancia": "Manuel",
"Pareja": "Luis, 60 años juntos",
"Ciudad actual": "Barcelona",
"Siempre ha vivido en la misma localidad": "Sí",
"Hijos": "Antonio (60), Marta (58), recuerdo de cuando cocinábamos
juntos los domingos.",
"Nietos": "Carlos (35), Laura (32), recuerdo de las tardes en el parque.",
"Mascotas": "Rex, perro",
"Amigos": "Antonio, Rosa",
"Estudios": "EGB, Bachillerato, Secretariado, diversos cursos de idiomas",
"Otra cosa que le hubiera gustado estudiar": "Historia del Arte",
"Anécdota de juventud": "Viajé en tren por toda Europa con mis amigos durante
un verano inolvidable.",
"Ocio en la juventud": "Bailar, tocar el piano",
"Profesión": "Secretaria",
"Coches en la vida": "Seat 600, Renault 5",
"Religiosa": "Sí",
"Religión": "Católica",
"Rutinas diarias": "Aseo personal, cuidado de la salud",
"Deporte de joven": "Natación",
"Deporte actual": "Paseos diarios",
"Deporte que le gusta ver": "Tenis",
```

```
"Lectura": "Novelas históricas, periódico local",
"Programas de televisión": "Concursos y películas",
"Ir al cine": "Sí",
"Juegos de mesa": "Cartas y dominó",
"Aficiones": "Jardinería, pintura",
"Vacaciones preferidas": "Playa y extranjero",
"Actividades con pareja": "Viajar y disfrutar de la ópera",
"Comida favorita": "Paella",
"Bebida favorita": "Vino tinto",
"Color favorito": "Azul",
"Complemento de ropa": "Bufanda",
"Adornos personales": "Pendientes y pulseras",
"Relajación": "Estar con familia y escuchar música",
"Cosas que no soporta": "Ruido excesivo y olores fuertes",
"Mayor logro en la vida": "Mantener una familia unida",
"Actividad no realizada que deseaba": "Aprender a bailar flamenco",
"Actividad que haría de otra manera": "Habría viajado más cuando era joven",
"Deseo para el resto de su vida": "Seguir disfrutando
de los momentos con sus seres
queridos",
"Deseo por cumplir": "Visitar Japón"
}

>>>** María: Una vida llena de recuerdos y amor**

María, cariñosamente llamada "Marucha" por su familia, nació un
soleado 15 de febrero de 1930 en la vibrante ciudad de Barcelona,
España. Su infancia transcurrió en las calles de la ciudad, donde
creció rodeada del amor de sus padres, Joaquín y Carmen, y sus dos
hermanos, Pedro y Ana.

Un día, mientras jugaba en el parque, la pequeña María se alejó
demasiado y se perdió. Afortunadamente, un amable policía la
encontró y la ayudó a regresar con sus padres, quienes la abrazaron
con alivio y alegría. Este evento quedó grabado en su memoria como
un recordatorio de la importancia de la familia y la amabilidad
de los extraños.

Su mejor amigo de la infancia fue Manuel, con quien compartía horas
de juegos y aventuras. Juntos exploraban la ciudad, descubrían nuevos
lugares y forjaban una amistad que duraría toda la vida.

En su juventud, María conoció a Luis, el hombre que se convertiría en
su compañero de vida. Se enamoraron perdidamente y se casaron,
iniciando un viaje juntos que duraría 60 años llenos de amor, risas
y complicidad. Formaron una hermosa familia, con dos hijos,
```

Antonio y Marta, quienes llenaron su hogar de alegría y bullicio.

María siempre había vivido en Barcelona, una ciudad que amaba con todo su corazón. Disfrutaba de pasear por sus calles, admirar su arquitectura y empaparse de su vibrante cultura. Barcelona era su hogar, el lugar donde había vivido toda su vida y donde había construido sus más preciados recuerdos.

Su vida profesional estuvo marcada por su trabajo como secretaria, una profesión que desempeñó con dedicación y responsabilidad. Además de su trabajo, María era una mujer apasionada por el aprendizaje. Completó sus estudios de EGB y Bachillerato, y luego se especializó en Secretariado. Además, siempre mostró interés por otros idiomas, tomando diversos cursos para ampliar sus conocimientos.

Entre sus aficiones favoritas se encontraban la jardinería y la pintura. Disfrutaba cuidando de sus plantas y flores, llenando su hogar de vida y color. También le encantaba pintar, plasmando en sus lienzos los paisajes y momentos que más la inspiraban.

Los domingos por la mañana, María y Luis tenían una tradición inquebrantable: cocinar juntos. Preparaban deliciosos platos, como la paella, su comida favorita, y disfrutaban de largas conversaciones mientras compartían la mesa con su familia. Estos momentos eran para ella los más preciados, llenos de risas, amor y el aroma de la comida casera.

En su tiempo libre, María también disfrutaba de otras actividades como leer novelas históricas, ver concursos y películas en la televisión, ir al cine, jugar a cartas y dominó con sus amigos Antonio y Rosa, y pasear por el parque con sus nietos, Carlos y Laura.

Su fe era una parte importante de su vida. Como católica devota, encontraba consuelo y fortaleza en su religión. Las rutinas diarias de aseo personal y cuidado de la salud eran esenciales para ella, ya que le permitían mantenerse activa y disfrutar de la vida al máximo.

A lo largo de su vida, María ha logrado grandes cosas. El mayor de sus logros, sin duda alguna, ha sido mantener una familia unida y llena de amor. Ha sido una madre ejemplar, una esposa dedicada y una

```
amiga incondicional. A pesar de que tenía algunas actividades
pendientes, como aprender a bailar flamenco y viajar más cuando era
joven, no hay duda de que ha vivido una vida plena y feliz.

En la actualidad, María sigue disfrutando de la vida al máximo. Su
mayor deseo es seguir compartiendo momentos con sus seres queridos y
atesorando cada instante. También anhela cumplir su último sueño:
visitar Japón, un país que siempre le ha fascinado.

María es un ejemplo de mujer fuerte, resiliente y llena de amor. Su
historia es un recordatorio de que la vida es un regalo precioso que
debemos aprovechar al máximo, llenándola de momentos felices, amor y
pasión.
```

5.2.3. Módulo de procesamiento de imágenes

El módulo de procesamiento de imágenes se encarga de extraer información de las mismas para generar nuevas preguntas y tratar de obtener información adicional. Las imagenes pueden ayudar al paciente a recordar, o expresarse y las preguntas específicas acerca de la imagen son más fáciles de responder que una pregunta genérica sin ningún apoyo visual.

Una vez llega una imagen al manejador, este la envía apropiadamente a este módulo y a su vez, el módulo llama al modelo *gemini-pro-vision* para que describa la misma y genere una pregunta. La respuesta a esa pregunta se analizará por el módulo de procesamiento de texto como cualquier otra respuesta.

En el caso de la figura 5.3 vemos un ejemplo de como el chatbot genera información a partir de una imagen y formula una pregunta específica acerca de la misma. De esta forma, la respuesta asociada a esa pregunta pasa a analizarse como cualquier otra respuesta.

5.3. Interfaz e interacción con el usuario

A pesar de que se podría utilizar en su versión por consola, el chatbot esta pensando para ser usado a través de Telegram. Gracias a esto puede estar disponible en todos aquellos dispositivos en los que es posible utilizar Telegram, entre los que destacaríamos los dispositivos móviles y los ordenadores.

Para poner en marcha la versión por ordenador, es necesario la aplicación de Telegram, ya sea la versión web o la versión local. De igual forma la versión en móvil o tablet requiere la aplicación de Telegram. Una vez con la aplicación correctamente instalada y abierta, hay que buscar al usuario @*mavice*07_*bot* y enviado el comando *start*, recibiremos el mensaje de bienvenida y comenzará la conversación.

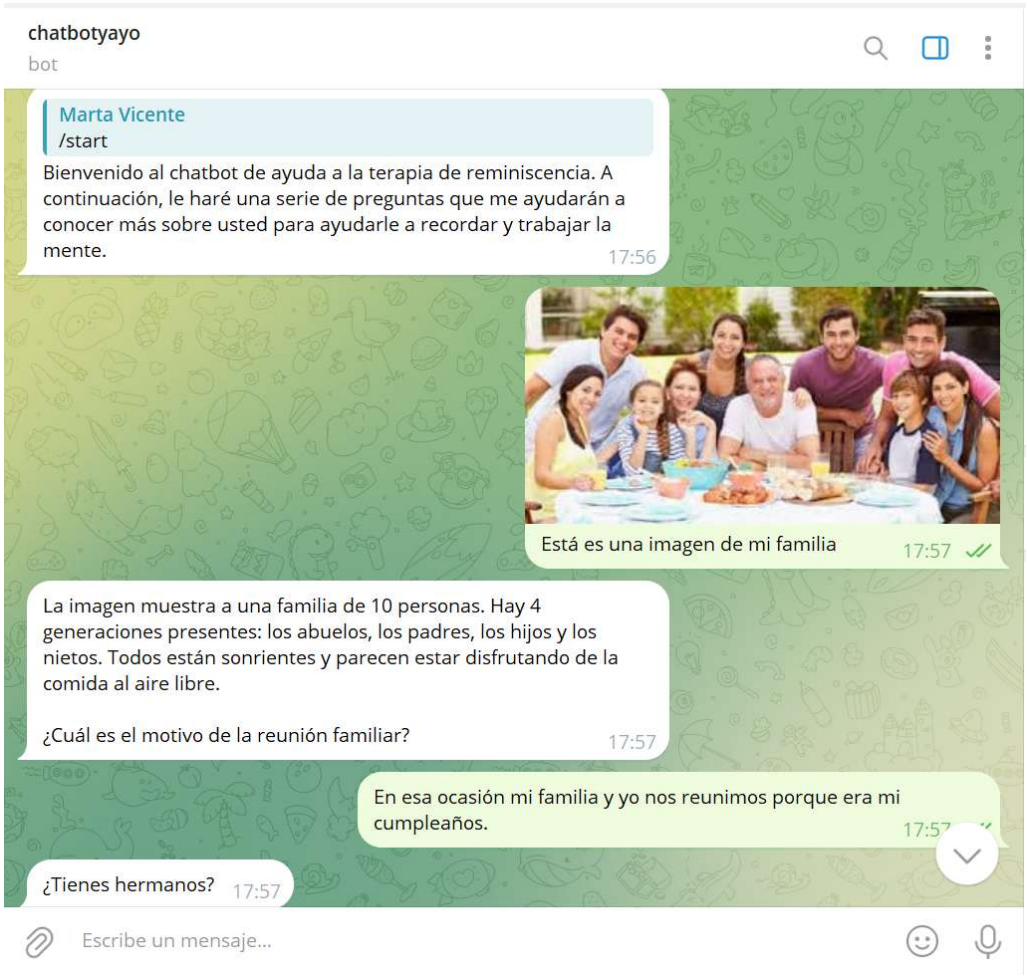

Figura 5.3: Ejemplo de extracción de información a partir de imagenes

Ejemplos de esto en la versión de ordenador y móvil se pueden ver respectivamente en las figuras 5.4 y 5.5.

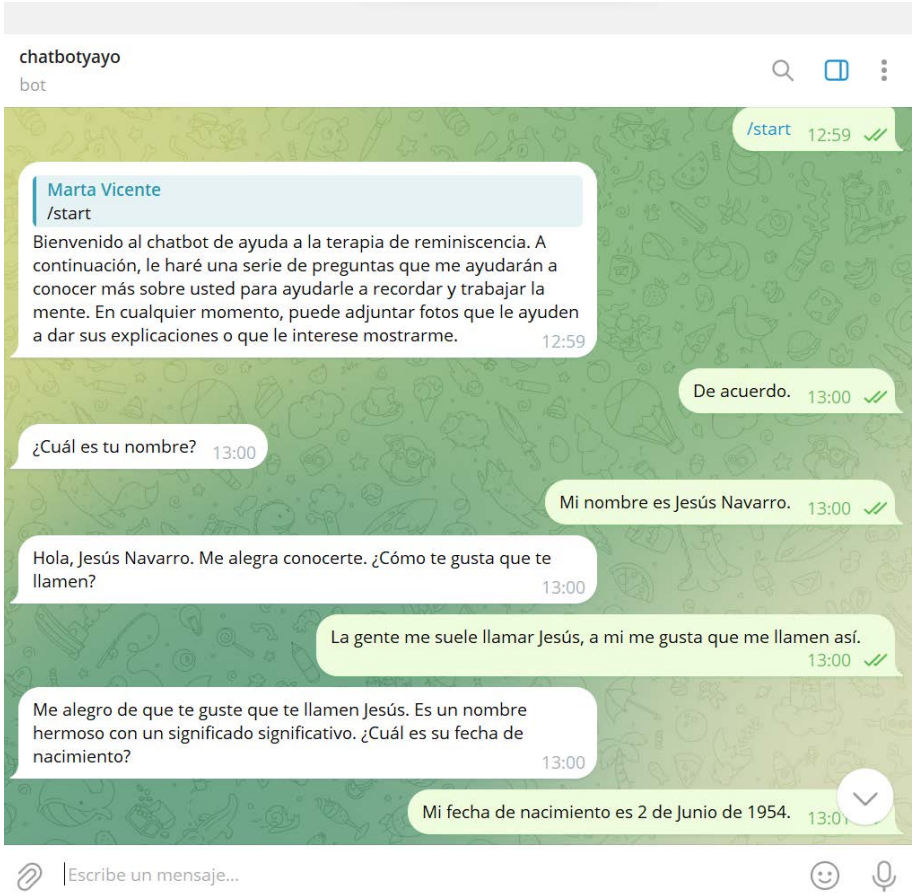

Figura 5.4: Ejemplo de bienvenida y primeras interacciones con la versión en ordenador

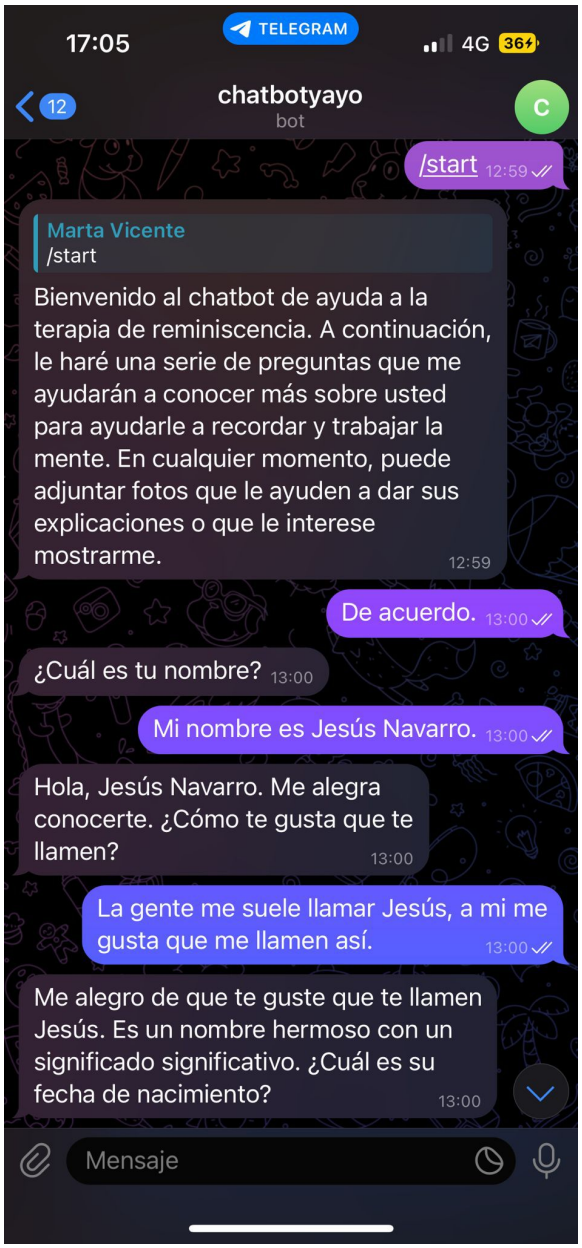

Figura 5.5: Ejemplo de bienvenida y primeras interacciones con la versión en móvil

Capítulo 6

Conclusiones y Trabajo Futuro

En el presente capítulo se describen las conclusiones a las que se ha llegado tras meses de trabajo en un proyecto de construcción de una herramienta conversacional de ayuda a la terapia de reminiscencia.

Pese al desarrollo exponencial que esta sufriendo la IA en el campo del procesamiento del lenguaje, la provisionalidad de los módelos y los constantes cambios en las legislaciones de los territorios han ocasionado diversos problemas y en consecuencia a un estudio más extenso de la situación actual y de las diferentes alternativas.

En este capítulo, no solo se pretende exponer las conclusiones acerca de la herramienta desarrollada, si no todas las conclusiones asociadas al estudio teórico previo.

Por otro lado, se describirán los posibles trabajos futuros que pueden considerarse fruto de la investigación.

6.1. Resultado final

Como resultado del desarrollo de este proyecto, se ha creado una herramienta conversacional para apoyar la terapia de reminiscencia.

La herramienta consiste en un chatbot capaz de mantener una conversación con el usuario, con el propósito de obtener tanta información del mismo como sea posible.

La herramienta es capaz de identificar qué información no se ha obtenido y generar preguntas coherentes en el contexto de la conversacióny así obtener los datos que faltan. Además, genera *feedback* que hace del chat una conversación con más sentido.

En cualquier momento se pueden enviar imagenes al chatbot de forma que se obtiene una descripción de las mismas y una pregunta asociada. Esto permite que el usuario pueda recordar cosas con mayor facilidad al hacerse preguntas más específicas sobre una imagen que tiene delante.

La interfaz de usuario resulta intuitiva y fácil de usar. El hecho de que este im-

plementada a través de Telegram reduce significativamente la curva de aprendizaje y la hace mucho más usable para todo tipo de usuarios. Al no requerir ningún conocimiento tecnológico especial esta herramienta es accesible no solo para terapeutas, sino también para familiares y pacientes que estén familiarizados con dispositivos móviles y computadoras.

Aunque inicialmente diseñada como una herramienta terapéutica para recopilar información sobre la vida del paciente, esta herramienta tiene múltiples usos. Por ejemplo, las versiones portátiles, como la aplicación móvil o para tabletas, permiten su uso como entretenimiento durante viajes largos que pueden resultar amenos para los usuarios. La facilidad de uso también permite que los pacientes la empleen en momentos de soledad, manteniendo conversaciones interesantes sobre su vida que estimulan la mente y les proporcionan compañía virtual.

La principal función de este bot es buscar y almacenar información relevante para generar historias de vida utilizando otras herramientas. Además, al final de la conversación se genera una historia de vida construida con toda la información que ha sido obtenida. Este resultado, supone una forma de *feedback* y una recompensa que mantiene la motivación del usuario para llegar hasta el final de la conversación.

6.2. Trabajo futuro

Como punto final de este trabajo se exponen aquellas posibles líneas abiertas de desarrollo e investigación, que han surgido tras el análisis del proyecto, y que podrían ser estudiados con la finalidad de mejorar el proyecto.

En primer lugar, aunque este trabajo genere una pequeña historia de vida como resultado final de la conversación sería interesante conectar este trabajo con el proyecto explicado en la sección 2.7.1.1 “Generación de historias de vida usando técnias de Deep Learning”. Para ello habría que transformar la información que obtiene el chatbot en un conjunto de entrada con el formato adecuado para su procesamiento. De esta forma, se generarían historias de vida más fieles a la vida real de la persona, y se tendrían herramientas para realizar todo el trabajo necesario asociado a la terapia de reminiscencia.

Por otro lado, otra modificación que podría resultar interesante sería la adaptación del modelo a múltiples lenguajes de forma que se amplié el público objetivo de la aplicación. La API de Gemini esta disponible en diferentes lenguajes con lo que podría mantenerse el uso de esta herramienta. Sin embargo, pero el manejo de todas las preguntas predefinidas tendría que ser traducido para la interacción con el usuario.

Puede ser interesante considerar la alternativa del reconocimiento de voz de forma que además de la interfaz de Telegram, se pudiera desarrollar una interfaz de voz donde el paciente pudiera hablar y escuchar como si se tratará de una llamada telefónica. Para ello se podrían adaptar algunas funcionalidades de modelos de Gemini. o bibliotecas como “Whisper” de OpenAI.

El desarrollo de la API de Gemini, y el acceso de nuevas versiones traerá consigo nuevas posibilidades. Entre ellas, destaca la integración de vídeos para enviar al chatbot y que se analicen con el objetivo de obtener más información bibliográfica.

Conclusions and Future Work

This chapter describes the conclusions reached after months of work on a project to build a conversational tool to support reminiscence therapy.

Despite the exponential development of AI in the field of language processing, the provisionality of models and constant changes in legislation across different regions have caused various issues, leading to a more extensive study of the current situation and different alternatives.

In this chapter, we aim not only to present the conclusions about the developed tool but also to share all conclusions associated with the prior theoretical study.

Furthermore, the potential future work that can be considered as a result of the research will be described.

6.3. Final Result

As a result of this project, a conversational tool has been created to support reminiscence therapy.

The tool consists of a chatbot capable of maintaining a conversation with the user, aiming to gather as much information as possible.

The tool can identify which information has not been obtained and generate coherent questions within the context of the conversation to gather the missing data. Additionally, it provides feedback that makes the chat a more meaningful conversation.

Images can be sent to the chatbot at any time, allowing it to provide a description and an associated question. This helps the user recall things more easily by answering more specific questions about an image they have in front of them.

The user interface is intuitive and easy to use. The fact that it is implemented through Telegram significantly reduces the learning curve and makes it much more usable for all types of users. Since it does not require any special technological knowledge, this tool is accessible not only to therapists but also to family members and patients who are familiar with mobile devices and computers.

Although initially designed as a therapeutic tool to gather information about the patient's life, this tool has multiple uses. For example, portable versions, such as a mobile or tablet application, allow it to be used as entertainment during long journeys, which can be enjoyable for users. Its ease of use also allows patients to use it during moments of loneliness, having interesting conversations about their life that stimulate their mind and provide virtual companionship.

The main function of this bot is to seek and store relevant information to generate life stories using other tools. Additionally, at the end of the conversation, a life story constructed with all the obtained information is generated. This result serves as feedback and a reward that keeps the user motivated to complete the conversation.

6.4. Future Work

Por supuesto, aquí tienes el texto traducido al inglés:

As a final point of this work, we present potential avenues for further development and research that have emerged from the project analysis and could be explored to enhance the project.

Firstly, although this work generates a small life story as the final result of the conversation, it would be interesting to connect this work with the project explained in section 2.7.1.1 "Generation of Life Stories Using Deep Learning Techniques." To achieve this, the information obtained by the chatbot would need to be transformed into an input set with the appropriate format for processing. This way, life stories more faithful to the person's real life would be generated, and tools would be available to carry out all the necessary work associated with reminiscence therapy.

Furthermore, another modification that could be interesting is adapting the model to multiple languages to expand the target audience of the application. The Gemini API is available in different languages, which would allow the continued use of this tool. However, all predefined questions for user interaction would need to be translated.

It may be interesting to consider the alternative of voice recognition so that, in addition to the Telegram interface, a voice interface could be developed where the patient can speak and listen as if it were a phone call. For this, some functionalities of the Gemini models or libraries such as OpenAI's "Whisper" could be adapted.

The development of the Gemini API and the access to new versions will bring new possibilities. Among them, the integration of videos to be sent to the chatbot for analysis to obtain more biographical information stands out.

Bibliografía

Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F. L. y Almeida, D. Gpt-4 technical report. *arXiv*, Disponible en `https://arxiv.org/abs/2303.08774`.

Aguilera Heredero, P. y Molina Muñoz, C. Extracción de información personal a partir de redes sociales para la creación de un libro de vida. 2021. Trabajo de fin de grado.

Barquilla Blanco, C., Díez García, P., Marco Mulas López, S. y Verdú Rodríguez, E. Recuérdame 1.0: Aplicación de apoyo para el tratamiento de personas con problemas de memoria mediante terapias basadas en reminiscencia. 2022. Trabajo de Fin de Grado, Universidad Complutense de Madrid.

Bird, S., Klein, E. y Loper, E. *Natural Language Processing with Python*. O'Reilly Media, Inc., 2009. ISBN 9780596516499.

Boto, A., Portillo Torres, A., Sanz Mayo, D. y Portillo Torres, R. *Extracción de preguntas a partir de imágenes para personas con problemas de memoria mediante técnicas de Deep Learning*. Proyecto Fin de Carrera, Facultad de Informática, Universidad Complutense de Madrid, 2021.

Cañete, J., Chaperon, G., Fuentes, R., Ho, J.-H., Kang, H. y Pérez, J. Spanish pretrained bert model and evaluation data. En *PML4DC at ICLR 2020*. 2020.

Champin, P.-A. Rdf tutorial. *arxiv*, 2002.

Chung, J. A. y Cummings, J. L. Neurobehavioral and neuropsychiatric symptoms in alzheimer's disease: characteristics and treatment. *Neurologic clinics*, vol. 18, páginas 829–846, 2000.

Comisión Europea. La comisión se congratula del acuerdo político sobre la ley de inteligencia artificial. Comunicado de prensa, 2023.

Cotelli, M., Manenti, R. y Zanetti, O. Reminiscence therapy in dementia: a review. *Maturitas*, vol. 72, páginas 203–205, 2012.

Devlin, J., Chang, M.-W., Lee, K. y Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. En *NAACL*. 2019.

Donoso, A. La enfermedad de alzheimer. *Revista Chilena de Neuro-psiquiatría*, vol. 41(supl.2), 2003. ISSN 0717-9227. ARTÍCULO ORIGINAL.

Díez Sobrino, C., Guerrero Sosa, E., Prieto Campo, A., Martínez González, P. y Arias Rodríguez-Peña, S. Recuérdame 2.0: Mejorando una aplicación de apoyo para el tratamiento de personas con problemas de memoria mediante terapias basadas en reminiscencia. 2023. Trabajo de Fin de Grado, Universidad Complutense de Madrid.

Edwards, B. Why chatgpt and bing chat are so good at making things up. *Ars Technica*, 2023.

ElDerecho.com. Ley de ia: plazos y fomento de la inversión en europa. 11-04-2024.

Huang, H.-C., Chen, Y.-T., Chen, P.-Y., Hu, H.-L. S., Liu, F., Kuo, Y.-L. y et al. Reminiscence therapy improves cognitive functions and reduces depressive symptoms in elderly people with dementia: a meta-analysis of randomized controlled trials. *Journal of the American Medical Directors Association*, vol. 16, páginas 1087–1094, 2015.

Irazoki, E., García-Casal, J. A., Sánchez-Meca, J. y Franco-Martín, M. Eficacia de la terapia de reminiscencia grupal en personas con demencia. revisión sistemática y metaanálisis. *Revista de neurología*, 2017.

Ji, Z., Lee, N., Frieske, R., Yu, T., Su, D., Xu, Y., Ishii, E. y Bang, Y. Survey of hallucination in natural language generation. *ACM Computing Surveys*, vol. 55(12), páginas 1–38, 2022.

Kudo, T. y Richardson, J. Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. *arXiv preprint arXiv:1808.06226*, 2018.

Leswing, K. Microsoft's bing a.i. made several factual errors in last week's launch demo. *CNBC*, 2023.

Lin, C. How to easily trick openai's genius new chatgpt. *Fast Company*, 2022.

Machin-Mastromatteo, J. D. Implicaciones y políticas editoriales de la inteligencia artificial. *arxiv*, 2023.

Magaz, L. L. Grado en ingeniería informática. 2023. Curso 2022-2023.

Mikolov, T., Sutskever, I., K. Chen, G. C., y Dean, J. istributed representations of words and phrases and their compositionality. *arXiv*, Disponible en `https://doi.org/10.48550/arXiv.1310.4546`.

Radford, A., Narasimhan, K., Salimans, T. y Sutskever, I. Improving language understanding by generative pre-training. OpenAI, 2018. Archived (PDF) from the original on 26 January 2021. Retrieved 23 January 2021.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D. y Sutskever, I. Language models are unsupervised multitask learners. 2019.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W. y Liu, P. J. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research 21 (2020) 1-67*, vol. 2020, Disponible en `https://jmlr.org/papers/volume21/20-074/20-074.pdf`.

Rajasekharan, A. A review of bert based models. 2019.

Rohrbach, A., Hendricks, L. A., Burns, K., Darrell, T. y Saenko, K. Object hallucination in image captioning. *UC Berkeley, Boston University*, 2023.

Rothman, D. *Transformers for Natural Language Processing*. Packt Publishing, 2nd edición, 2022.

Sage. Chatgpt and generative ai: Use of large language models and generative ai tools in writing your submission. Online, 2023.

Salas, C. A. Generacion de historias de vida usando técnicas de deep learning. Disponible en `https://docta.ucm.es/entities/publication/28458f98-f632-4b80-a883-a4c3169f9dbd`.

da Silva, P. P., Sugiura, N., Simperl, E., Wallace, E. y Han, H. 6th international semantic web conference, 2nd asian semantic web conference, ISWC 2007 + ASWC 2007, busan, korea, november 11-15, 2007, proceedings. 2007.

Slovikovskaya, V. Transfer learning from transformers to fake news challenge stance detection (fnc-1) task. *arXiv preprint arXiv:1910.14353*, 2019.

Spinak, E. ¿es que la inteligencia artificial tiene alucinaciones? *SciELO en Perspectiva*, 2023.

Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., Rodriguez, A., Joulin, A., Grave, E. y Lample, G. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

Tu, X., Lai, K. y Yanushkevich, S. Transfer learning on convolutional neural networks for dog identification. En *2018 IEEE 9th International Conference on Software Engineering and Service Science (ICSESS)*, páginas 357–360. IEEE, 2018.

UK, D. *Life story template*. Packt Publishing, 2023.

Vaswani, A.Shazeer, N.Parmar, N.Uszkoreit, Jones, J., L.Gomez, A. N., K. y Polosukhin. Advances in neural information processing systems. *arxiv*, vol. 30, 2023.

Wang, A. y Cho, K. Bert has a mouth, and it must speak: Bert as a markov random field language model. En *Proceedings of the Workshop on Methods for Optimizing and Evaluating Neural Language Generation*, páginas 30–36. Association for Computational Linguistics, Minneapolis, Minnesota, 2019.

Wikipedia. Alucinación (inteligencia artificial) - wikipedia. Fecha de última modificación.

Yang, J., Jin, H., Tang, R., Han, X., Feng, Q., Jiang, H., Yin, B. y Hu, X. Harnessing the power of llms in practice: A survey on chatgpt and beyond. *arXiv*, 2023.

Zhu, Y., Kiros, R., Zemel, R., Salakhutdinov, R., Urtasun, R., Torralba, A. y Fidler, S. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. En *The IEEE International Conference on Computer Vision (ICCV)*. 2015.

Zhuo, T. Y., Huang, Y., Chen, C. y Xing, Z. Exploring ai ethics of chatgpt: A diagnostic analysis. 2023.