
Desarrollo de una herramienta basada en
lenguaje de apoyo a la terapia basada en
reminiscencia
A language-based tool 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 una herramienta basada en
lenguaje de apoyo a la terapia basada en
reminiscencia

A language-based tool 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

1 de mayo de 2024

Dedicatoria

A Mamá

Agradecimientos

Resumen

Desarrollo de una herramienta basada en lenguaje 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. Su llegada supone un cambio significativo en la vida de quienes la padecen y en su entorno. Actualmente, se estima que en España 900.000 personas sufren esta y otras formas de demencia, y se proyecta que los casos se duplicarán para el año 2050. Por lo tanto, es de vital importancia desarrollar técnicas que puedan ralentizar el avance de la enfermedad. Aunque no es reversible, existen terapias que pueden mejorar la calidad de vida tanto del paciente como de sus seres queridos.

La terapia de reminiscencia es una modalidad terapéutica que se enfoca en ayudar a las personas a recordar y compartir sus experiencias y recuerdos pasados, especialmente aquellos relacionados con eventos significativos en sus vidas. Aunque es comúnmente empleada con personas mayores, también puede resultar efectiva en otros grupos de edad.

Este proyecto se centra principalmente en el desarrollo de un ChatBot que interactúe con el paciente, recopilando la información necesaria para construir una historia de vida y las imágenes asociadas. Para lograrlo, se ha clonado la API de Gemini y se ha entrenado con datos personalizados, de manera que genere historias de vida específicas y pertinentes.

Este enfoque promete ser una herramienta valiosa para mejorar la calidad de vida de las personas afectadas por Alzheimer y demencias similares.

Palabras clave

Reminiscencia, Chatbot, historia de vida, alzhéimer

Abstract

A language-based tool to support reminiscence therapy

Alzheimer's disease is a neurodegenerative condition that affects cognitive functions, memory, thinking, and behavior. Its onset marks a significant change in the lives of those affected and their surroundings. Currently, it is estimated that 900,000 people in Spain suffer from this and other forms of dementia, and it is projected that cases will double by the year 2050. Therefore, it is crucial to develop techniques that can slow down the progression of the disease. Although it is not reversible, there are therapies that can improve the quality of life for both the patient and their loved ones.

Reminiscence therapy is a therapeutic approach focused on helping individuals remember and share their past experiences and memories, especially those related to significant events in their lives. While commonly used with older individuals, it can also be effective with other age groups.

This project primarily focuses on the development of a ChatBot that interacts with the patient, gathering the necessary information to construct a life story along with associated images. To achieve this, the Gemini API has been cloned and trained with customized data, enabling it to generate specific and relevant life stories.

This approach holds promise as a valuable tool for enhancing the quality of life for individuals affected by Alzheimer's and similar dementias.

Keywords

reminiscence, chatbot, life story, Alzheimer's

Índice

1. Introducción	1
1.1. Motivación	1
1.1.1. Enfermedad de Alzheimer	2
1.1.2. Terapias de reminiscencia	3
1.1.3. Historia de vida	3
1.2. Objetivos	4
1.3. Estructura de la memoria	4
Introduction	7
2. Estado de la Cuestión	9
2.1. Evolución del Procesamiento del Lenguaje Natural	10
2.2. Word embeddings	10
2.2.1. TF-IDF	11
2.2.2. Bolsa de palabras	11
2.2.3. Word2Vec	12
2.3. Transformers	13
2.3.1. Aprendizaje por transferencia	13
2.4. Modelos de Lenguaje de Gran Tamaño (LLM)	14
2.4.1. BERT	15
2.4.2. T5	16
2.4.3. GPT <i>Generative Pretrained Transformer</i>	16
2.4.4. Llama	20
2.4.5. LaMDA	20

2.5.	Otros trabajos relacionados	21
2.5.1.	Proyecto Cantor	21
2.5.2.	Celia	23
3.	Tecnologías utilizadas	25
3.1.	Bibliotecas de Procesamiento del Lenguaje en Python	26
3.1.1.	NLTK	26
3.1.2.	SpaCy	28
3.2.	APIs de procesamiento del lenguaje	30
3.2.1.	Bard	30
3.2.2.	Gemma	31
3.2.3.	GPT API	32
3.2.4.	Rasa	32
3.2.5.	Gemini	32
3.3.	Respuestas de Gemini	33
3.4.	Almacenamiento de la información	36
3.4.1.	JSON	36
3.4.2.	RDF	36
3.5.	Desarrollo de la interfaz	37
3.6.	Programación orientada a objetos	38
3.7.	VPN	39
4.	Desarrollo de prototipos	41
4.1.	Planteamiento del problema	41
4.2.	Preguntas predefinidas	42
4.3.	BARD	42
4.4.	Prototipo usando GEMMA	45
4.4.1.	Google Collaborate	45
4.4.2.	Linux	46
4.5.	Desarrollo de la interfaz con RASA	47
4.6.	Gemini	49
4.7.	Análisis de las respuestas	49
4.7.1.	Análisis de las respuestas y identificación de la información omitida	49

4.7.2. Generación de preguntas para obtener la información faltante	50
4.8. Interfaz e interacción con el usuario	50
4.8.1. Integración de Gemini y RASA	51
4.9. Extracción de información a partir de imágenes	51
5. Herramienta conversacional de ayuda a la terapia a la reminiscencia	53
5.1. Herramientas y puesta en marcha	53
5.1.1. VPN	53
5.2. Instalación de la API de Gemini	54
5.3. Arquitectura	54
5.3.1. Interfaz e interacción con el usuario	54
5.3.2. Almacenamiento y manejo de la información	54
5.3.3. Generación de historias de vida	54
6. Conclusiones y Trabajo Futuro	55
Conclusions and Future Work	57
Bibliografía	59

Índice de figuras

2.1. Desarrollo de los LLM por Yang et al. (2023)	14
2.2. Diagrama del modelo T5 de Raffel et al. (2020)	16
2.3. Arquitectura de decodificadores usada por GPT-2 Salas (2020)	18
3.1. Árbol sintáctico generado con nltk	27
3.2. Árbol sintáctico generado con nltk	29
3.3. Ejemplo de uso de BARD.	30
3.4. Ejemplo de uso de GEMMA.	31
3.5. Ejemplo de un recurso RDF.	37
3.6. Ejemplo de un recurso RDF SCHEMA.	38
3.7. Ejemplo de interfaz generada con PyQT5	39
4.1. Ejemplo de uso de BARD.	43
4.2. Muestra de la interacción chatbot-usuario en el primer prototipo	44
4.3. Ejemplo de uso de BARD.	48
4.4. Análisis de respuestas	50
4.5. Generación de preguntas	50
4.6. Ejemplo de extracción de información a partir de imágenes	52

Índice de tablas

Introducción

“Hay enfermos incurables, pero ninguno incuizable”
— Francesc Torralba

El avance en el campo de la medicina en el último siglo ha permitido un notable aumento en la esperanza de vida a nivel mundial. Sin embargo, este alargamiento de la vida también ha traído consigo un aumento en las enfermedades relacionadas con la vejez, como el Alzheimer. A medida que vivimos más tiempo, enfrentamos un mayor riesgo de desarrollar estas condiciones.

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.

La terapia de reminiscencia 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 la aplicación YayoBot 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

El Alzheimer, una enfermedad neurodegenerativa progresiva y devastadora, afecta a millones de personas en todo el mundo. Se caracteriza por la pérdida gradual de la memoria y otras funciones cognitivas, lo que eventualmente conduce a la incapacidad para llevar a cabo las actividades diarias más básicas. Su curso clínico se divide en varias etapas distintas, cada una con sus propias características y desafíos.

- **Etapas Temprana o Leve:**

En esta fase inicial, los síntomas pueden pasar desapercibidos o atribuirse a simples descuidos. La persona afectada puede experimentar dificultades para recordar nombres, eventos recientes o encontrar 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.

- **Etapas Intermedia o Moderada:**

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.

- **Etapas Avanzada o Severa:**

En esta etapa tardía, el Alzheimer alcanza su punto más devastador. 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 desarrollo del Alzheimer se asocia con cambios físicos y químicos en el cerebro, incluida la acumulación de placas de proteínas llamadas beta-amiloide y ovillos neurofibrilares compuestos de proteína tau. Estas alteraciones provocan la muerte de células nerviosas y la disrupción de las conexiones entre ellas, lo que resulta en la progresiva pérdida de funciones cognitivas y conductuales.

Aunque no existe cura para el Alzheimer, existen tratamientos farmacológicos y terapias no farmacológicas que pueden ayudar a aliviar los síntomas y mejorar la calidad de vida de los pacientes en las etapas tempranas y moderadas de la enfermedad. Sin embargo, a medida que avanza la enfermedad, el enfoque se centra más en la atención y el apoyo integral, tanto para la persona afectada como para sus cuidadores y familiares.

El Alzheimer es una enfermedad desgarradora que afecta no solo a quienes la padecen, sino también a sus seres queridos. La investigación continua es fundamental

para comprender mejor sus mecanismos subyacentes, desarrollar tratamientos más efectivos y, en última instancia, encontrar una cura para esta enfermedad que roba la memoria y la identidad de quienes la sufren.

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.

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, los 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

Este trabajo busca desarrollar un chatbot que permita a los terapeutas aplicar terapia de reminiscencia a sus pacientes. En concreto, se centra en la primera etapa de búsqueda de la información con el objetivo de, más adelante generar las historias de vida y aplicar la terapia.

Para cumplir este objetivo general, se abordan los siguientes objetivos específicos:

- Desarrollar un primer chatbot básico capaz de realizar preguntas predefinidas y almacenar las respuestas de forma eficiente.
- Estudiar las técnicas modernas de procesamiento del lenguaje como los LLMs, bibliotecas como NLTK y spaCy. Además, de las diferentes APIs de para el desarrollo de chatbots.
- Mejorar el primer chatbot haciendolo 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 del chatbot que sea capaz de analizar las repuestas y sepa tirar del hilo. También que sea capaz de generar preguntas adecuadas.
- Generar una interfaz sencilla de usar para que haga del chatbot una herramienta útil para el propósito para el que ha sido desarrollada.
- Ordenar la información obtenida para que sea fácil de analizar, entender y procesar, con la intención de generar historias de vida a partir de ella.

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 5 capítulos, incluyendo el presente capítulo. 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 Alzheimer y la terapia de remiscencia. También muestra otros trabajos relacionados, contexto y precedentes del proyecto.
- **Capítulo 3: Marco teórico** 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 del chatbot** 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: Versión final y resultados** En el capítulo 5, se presenta la interfaz 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** En el capítulo 5, se presenta la interfaz 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.
- **Anexos** Adicionalmente, en el anexo se muestra una conversación con un paciente real para ver el funcionamiento del chatbot.

Introduction

Cuando tenga la versión definitiva en Español, pasaré a inglés.

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 esta forma, nos permite entender su importancia y conocer otras alternativas de implementación.

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 surgimiento 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 semánti-

ca 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.

$\mathbf{tf-idf}_{i,j}$ = Frecuencia del término i en el documento j \times 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.

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

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.

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

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", "perro" y "juguete", y un documento tiene una frecuencia de dos para "gato", 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 tareas 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-ajutable, 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 líneas de desarrollo principales para estos modelos de lenguaje, y multitud de modelos como se puee ver en la figura 2.1.

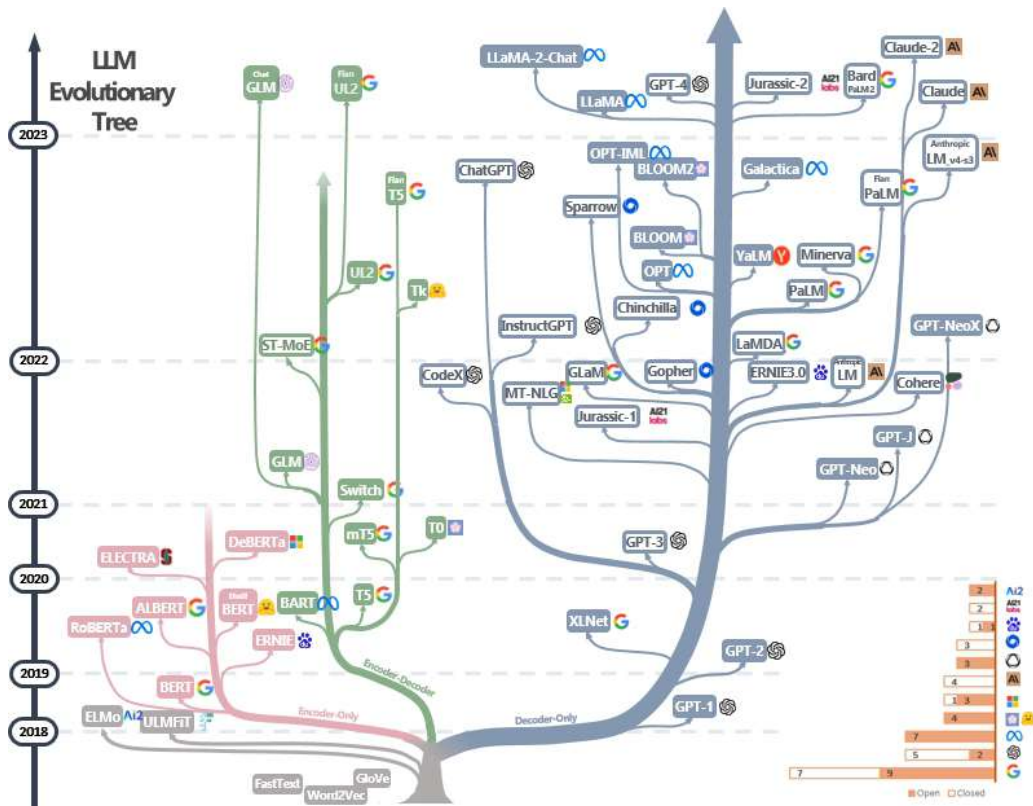


Figura 2.1: Desarrollo de los LLM por Yang et al. (2023)

Por un lado, el grupo “solo codificador”, mostrado en rosa en la figura 2.1 in-

cluye 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.

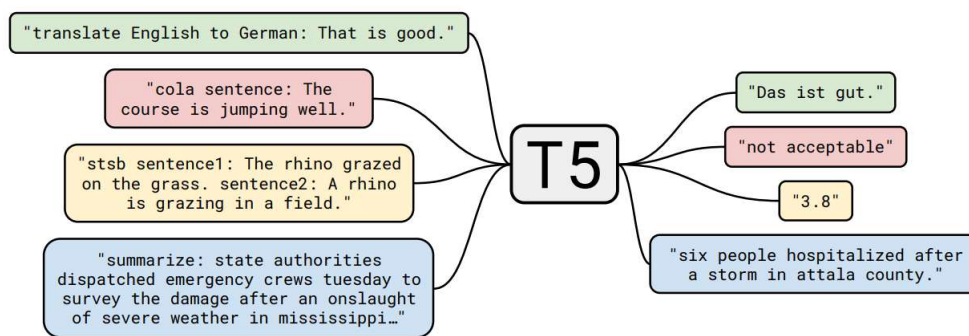


Figura 2.2: Diagrama del modelo T5 de Raffel et al. (2020)

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

2.4.3. GPT *Generative Pretrained Transformer*

Los modelos de lenguaje casual o *Casual Language Models*, tienen como idea principal predecir una palabra o token enmascarado en una oración. Podemos ver una palabra enmascarada como un hueco en blanco en una frase. En este caso, el modelo solo consideraría el contexto anterior (palabras situadas a su izquierda) y dejaría de tener en cuenta el contexto posterior. El sentido de procesamiento (izquierda a derecha o derecha a izquierda) no es relevante mientras que se realice en un único sentido, teniendo en cuenta para la predicción las palabras pertenecientes a ese contexto y dejando de lado el resto de palabras. Así, la propiedad clave de este tipo de modelos es la naturaleza unidireccional en su predicción de las palabras enmascaradas y se verá reflejado en el esquema de entrenamiento (Rothman, 2022).

2.4.3.1. GPT-1

Generative Pre-trained Transformer 1 (GPT-1) fue el primer modelo de lenguaje grande de OpenAI siguiendo la invención de la arquitectura transformer por parte de Google en 2017. En junio de 2018, OpenAI publicó un artículo titulado “Improving Language Understanding by Generative Pre-Training” Radford et al. (2018), en el cual presentaron ese modelo inicial junto con el concepto general de un *transformer* generativo pre-entrenado.

Hasta ese momento, los modelos neuronales de PLN con mejor rendimiento empleaban principalmente el aprendizaje supervisado a partir de grandes cantidades de datos etiquetados manualmente. Esta dependencia del aprendizaje supervisado limitaba su uso de conjuntos de datos que no estaban bien etiquetados, además de hacer que fuera prohibitivamente costoso y lento entrenar modelos extremadamente grandes; muchos idiomas (como el suajili o el criollo haitiano) son difíciles de traducir e interpretar utilizando tales modelos debido a la falta de texto disponible para la construcción de corpus. En contraste, el enfoque “semi-supervisado” de un GPT involucraba dos etapas: una etapa de pre-entrenamiento generativo no supervisado en la que se utilizaba un objetivo de modelado de lenguaje para establecer parámetros iniciales, y una etapa de ajuste fino discriminatorio supervisado en la que estos parámetros se adaptaban a una tarea objetivo.

El uso de una arquitectura transformer, en lugar de técnicas anteriores que involucraban RNNs con atención aumentada, proporcionó a los modelos GPT una memoria más estructurada de lo que se podría lograr a través de mecanismos recurrentes; esto resultó en un robusto rendimiento de transferencia en diversas tareas.

2.4.3.2. GPT-2

GPT-2 es uno de los modelos más representativos de los modelos de lenguaje casual que sigue una arquitectura de tipo Transformer basada en auto-atención (masked self-attention). Presentado por OpenAI en 2019, desde el primer momento fue considerado un gran éxito en el campo del Procesamiento de Lenguaje debido a sus más de 1.5 billones (americanos) de parámetros.

El objetivo de este sistema es construir una distribución de probabilidad en la que para cada palabra posible a generar se le asigna una probabilidad en función del contexto anterior. Este modelo fue pre-entrenado sobre un gran corpus de texto inglés siguiendo el entrenamiento auto-supervisado (self-supervised). El objetivo de este modelo es la predicción de la palabra siguiente dada una secuencia de palabras u oración.

Para pre-entrenar el modelo se creó un conjunto de datos, llamada WebText, formada a partir de la extracción de millones de páginas web obtenidas de enlaces de salida de páginas de Reddit que habían recibido una determinada puntuación mínima (para garantizar la calidad y significancia del enlace). Las páginas de Wikipedia relacionadas con estos enlaces se eliminaron. Es por esto por lo que GPT-2 no está entrenado bajo ningún texto de Wikipedia. El conjunto de datos resultante es un

enorme corpus de 40GB de textos preparado para el entrenamiento de este modelo, GPT-2 Radford et al. (2019).

La estructura de GPT-2 se asemeja mucho a la del modelo Transformer. Originalmente, este modelo constaba de un codificador y un decodificador, diseñado especialmente para tareas específicas como la traducción automática. Sin embargo, GPT-2 abandonó esta arquitectura convencional y optó por utilizar exclusivamente una serie de decodificadores del modelo Transformer. El número de decodificadores utilizados varía según el tamaño de GPT-2: la versión Small emplea solo doce, mientras que la versión Extra Large puede llegar a utilizar hasta cuarenta y ocho decodificadores. Esta configuración basada únicamente en decodificadores, característica distintiva de GPT-2, se ilustra en la figura 2.17.

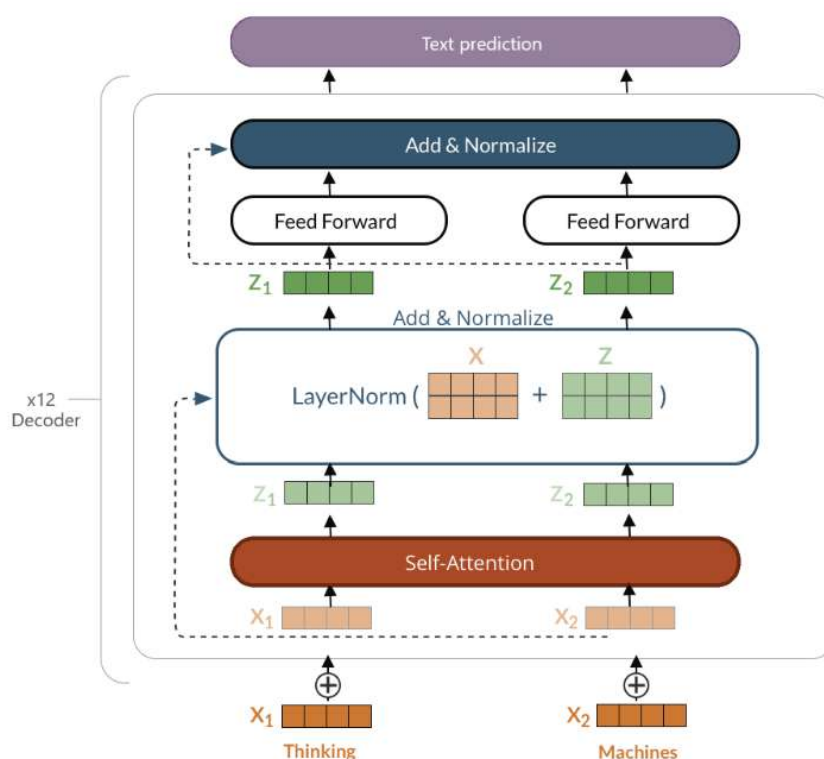


Figura 2.3: Arquitectura de decodificadores usada por GPT-2 Salas (2020)

GPT-2 es un modelo unidireccional que utiliza auto-regresión para generar texto. Se centra en la secuencia anterior a la última palabra y no considera la secuencia posterior. Genera una nueva palabra basada en la secuencia actual y la añade a la entrada para el siguiente paso. Sin embargo, tiene limitaciones, como sesgos en la generación debido al conjunto de datos y al algoritmo de entrenamiento. Además, solo está disponible en inglés, aunque se puede usar Google Translate para otros idiomas. A pesar de esto, los modelos de lenguaje como GPT-2 son útiles para generar texto fluido y completo, prediciendo palabras de un corpus basándose en el contexto anterior.

2.4.3.3. GPT-3

GPT-3 aumentó significativamente la cantidad de parámetros en comparación con GPT-2, alcanzando hasta 175 mil millones de parámetros, lo que permitió una mayor complejidad y capacidad de aprendizaje. Debido a su mayor capacidad y tamaño de conjunto de datos, GPT-3 demostró una mejor capacidad 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.

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

GPT-3 produjo textos con una mayor precisión y coherencia, lo que resultó 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 según el informe técnico publicado por la propia compañía (Achiam et al., 2023), es el siguiente modelo de gran escala de OpenAI capaz de aceptar entradas de imagen y texto y 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 alrededor del 10 % superior de los examinados. GPT-4 es un modelo basado en *Transformer* 2.3 preentrenado para predecir el siguiente token en un documento. El proceso de alineación posterior al entrenamiento resulta en un mejor rendimiento en medidas de factualidad y adherencia al comportamiento deseado.

Una de las características principales de GPT-4 es su capacidad para procesar entradas de texto y de imagen y producir salidas de texto. Se evaluó en una variedad de exámenes diseñados originalmente para humanos, donde obtuvo resultados destacados, superando en muchos casos a la mayoría de los examinados humanos. Por ejemplo, en un examen simulado de abogacía, GPT-4 obtiene una puntuación que se encuentra en el 10 % superior de los examinados, en contraste con GPT-3.5, que se encuentra en el 1 % inferior.

En una serie de benchmarks tradicionales de NLP, GPT-4 supera tanto a modelos de lenguaje grandes anteriores como a la mayoría de los sistemas de vanguardia (que a menudo tienen 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. Se debe tener cuidado al usar las salidas de GPT-4, especialmente en contextos donde la fiabilidad es importante. Las

capacidades y limitaciones de GPT-4 plantean desafíos de seguridad significativos y novedosos, y se considera que el estudio cuidadoso de estos desafíos es un área de investigación importante dada la posible repercusión societal.

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.

2.4.5. LaMDA

LaMDA, *Language Model for Dialogue Applications* (Modelo de Lenguaje para Aplicaciones de Diálogo), es una familia de modelos de lenguaje neuronal conversacional desarrollados por Google.

¹El 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.

La primera generación se anunció en la Google I/O ² de 2021 y fue presentada como un modelo de lenguaje conversacional. La segunda generación, presentada en el evento del año siguiente, introdujo mejoras y nuevas capacidades, como la generación de conversaciones originales sobre temas no enseñados previamente.

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

Las habilidades conversacionales de LaMDA tardaron años en desarrollarse. Como muchos modelos de lenguaje recientes, incluidos BERT y GPT-3, se basa en *Transformer* 2.3. Esa arquitectura produce un modelo que se puede entrenar para leer muchas palabras (una oración o párrafo, por ejemplo), preste atención a cómo esas palabras se relacionan entre sí y luego prediga qué palabras cree que vendrán a continuación.

Pero a diferencia de la mayoría de los otros modelos de lenguaje, LaMDA fue entrenado en diálogo. Durante su formación, captó varios de los matices que distinguen la conversación abierta de otras formas de lenguaje. Uno de esos matices es la sensatez.

El modelo de Transformer que utiliza LaMDA es de solo decodificador y está pre-entrenado en un corpus de texto que incluye documentos y diálogos. Se entrenó con datos de ajuste fino y se probó en diferentes modelos con diferentes hiperparámetros, demostrando superar las respuestas humanas en áreas específicas.

2.5. Otros trabajos relacionados

2.5.1. 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.

²Google I/O es una conferencia anual de desarrolladores realizada por Google en Mountain View, California

2.5.1.1. Generación de historias de vida usando técnicas 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 bibliográfica que da lugar a las historias de vida.

2.5.1.2. 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.

2.5.1.3. 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, una técnica terapéutica conocida como terapia basada en reminiscencia.

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.

2.5.1.4. 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).

2.5.1.5. Recuerdame 1.0

Recuerdame 2.0 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.

2.5.1.6. 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.

2.5.2. Celia

Celia,³ 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.

³Así es Celia, la inteligencia artificial para adultos mayores que puede detectar indicios de alzheimer

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.

Capítulo 3

Tecnologías utilizadas

El capítulo actual se enfoca en detallar 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 comprensible 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.

```
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

¹Página oficial de NLTK

```
>>>[('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.

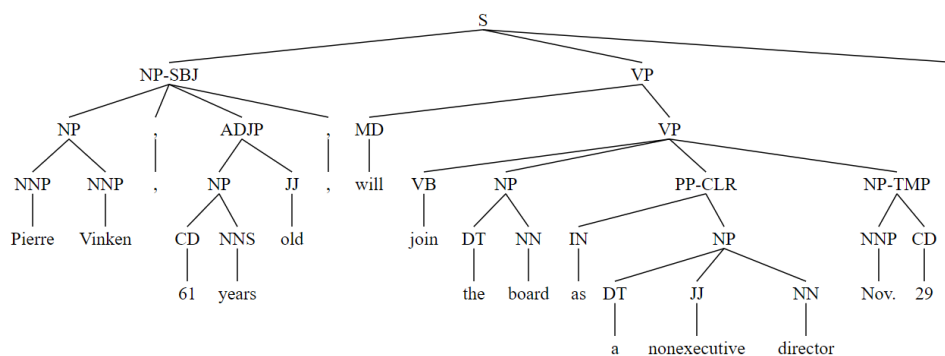


Figura 3.1: Árbol sintáctico generado con nltk

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

```

Listing 3.3: Análisis sintáctico y lematización 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
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

²spaCy

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 escuchando 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 *attribute_ruler* indica proporciona reglas de etiquetado a *tagger*.

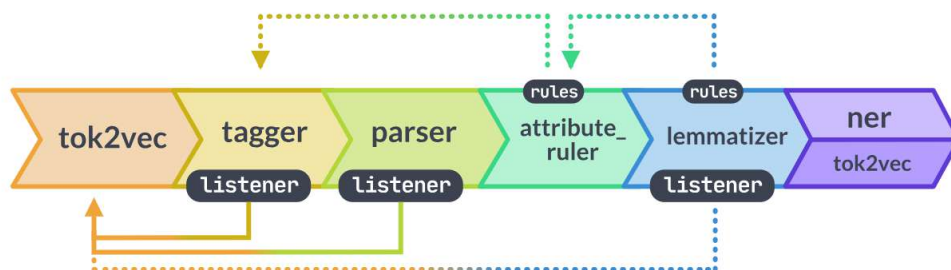


Figura 3.2: Árbol sintáctico generado con nltk

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 presentan 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, 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?
```

Figura 3.3: Ejemplo de uso de BARD.

Cómo 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 opción ideal 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 generación 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 ventajas 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 genera respuestas adecuadas y coherentes. En concreto, una respuesta generada por *gemma-7b* sería la siguiente. Sin embargo, las limitaciones propias de Google Collaborate no

```
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.
```

Figura 3.4: Ejemplo de uso de GEMMA.

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, clasificación de texto, y más. GPT API 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 ventajas de GPT API es su rendimiento en tareas de generación de texto, donde ha establecido nuevos estándares de calidad y coherencia en muchos casos. Además, GPT API 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, GPT API puede requerir recursos computacionales significativos y puede ser 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. Sin embargo, aunque rasa no es la API más potente en cuanto a generación de texto, tiene numerosas aplicaciones que resultan interesantes. Por ejemplo, gracias a la api de rasa es fácil volcar la interfaz de código en Python sobre la interfaz de Telegram.

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.

Para configurarlo, en primer lugar hay que ejecutar el programa `mibot.py` estando en telegram en la conversaa conversación de telegram el comando `/start` para comenzar la conversación y ya.

Sin embargo, y pese a las grandes funcionalidades de todas estas alternativas nos hemos decantado por hacer una interfaz con telegram desarrollando nuestro propio chatbot usando rasa.

Para desarrollar el chatbot de telegram me he decantado por usar la interfaz de telegram para la cuál se necesita la API de Rasa. Las principales ventajas que ofrece esta herramienta es la facilidad del manejo de la interfaz pues telegram es una herramienta muy conocida con la que los terapeutas pueden estar más familiarizados. Además, esto nos permite también usar la versión del chatbotyayo para móvil.

Para crear está interfaz hay que: 1. Instalar rasa 2. Instalar telegram 2. Obtener una api de rasa 4. crear un nuevo chatbot desde telegram con `@botFather` 5. Enviar a tu chatbot el comando `/start`

Una vez seguidos todos estos pasos ya puedes comenzar a interactuar con la API de rasa.

3.3. Respuestas de Gemini

El modelo más apropiado para el procesamiento de texto es *gemini – pro*. 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 3.6: 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. Ten 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.

3.4. Almacenamiento de la información

3.4.1. 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 '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.

3.4.2. 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 3.5. 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 3.5: 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 3.6 podemos ver un ejemplo de un RDF Schema.

3.5. 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 3.6: 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.

3.6. 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-

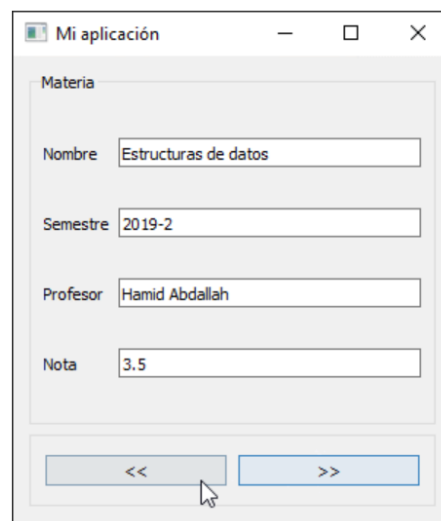


Figura 3.7: Ejemplo de interfaz generada con PyQt5

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.

3.7. VPN

Claro, aquí tienes un texto más detallado y técnico sobre qué son las VPN y sus ventajas:

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 encrip-

ta 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, con el estudio teórico llevado a cabo en el capítulo 3 y dando lugar al sistema que se detallará en el capítulo 5.

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

Cómo se comentó en el capítulo 1, el objetivo de este proyecto es construir una herramienta que facilite a los terapeutas la labor de realizar terapias de reminiscencia a personas con demencia. Para ello, se trata de obtener un chatbot capaz de realizar preguntas adecuadas, extraer información útil de ellas y generar nuevas preguntas para rellenar la información faltante.

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. 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, ofrece una serie de campos predefinidos que abarcan una amplia gama de aspectos relevantes. Dichos campos han sido seleccionados, con el objetivo de garantizar una exploración exhaustiva y completa de la vida del paciente tratando de asegurar que se aborden aspectos clave de la vida del paciente, 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 4.9). 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. BARD

El segundo hito en el desarrollo del chatbot fue mejorar el primer chatbot haciéndolo más inteligente y capaz de analizar las respuestas, identificar la información omitida y hacer preguntas específicas para obtener la información faltante.

Para el almacenamiento de la información se decidió usar *JSON* debido a su

estructura legible y su amplia compatibilidad, suponen una elección tecnológica ideal 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.

Por otro lado, se desarrollo una pequeña clase de forma provisional. Esta clase es la clase *Paciente* que tenía como objetivo estructurar y reunificar la información.

A continuación, se busco poder hacer un análisis de las respuestas exhaustivo que permitiera extraer información útil de las respuestas dadas por el usuario.

Tras el estudio realizado en la sección 3.2 se tomo la decisión de utilizar la API de BARD. Las principal característica que llevo 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 y creadores de contenidos, permitiendo la automatización de la generación de contenidos para diversas necesidades, como en este caso la generación de un chatbot. Todo esto, supone una gran ventaja frente a otras herramientas como las distintas bibliotecas que suponen desarrollar y generar nuestros propios algoritmos de generación del lenguaje, o otras APIs que requieren un entrenamiento que da lugar a horas y horas de computo para obtener resultados, a lo sumo, tan buenos como los que ofrecía BARD con su simple instalación.

Para instalar BARD se utilizo el entorno de *Google Collaborate*. En primer, lugar fue necesario instalar los paquetes *bardapi* y *pyTelegramBotApi*. Una vez intaladas las bibliotecas y para usar la API es necesario registrarse y obtener un token que nos sirve como identificador de la API.

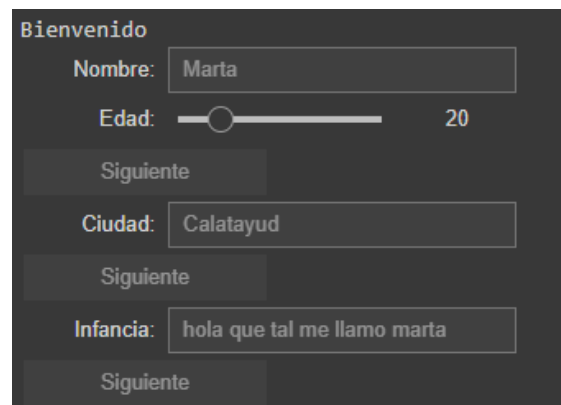
Una vez con todo instalado, se utilizó la API Bard para extraer información a partir de respuestas a preguntas específicas. Como se ve en la figura 4.1, BARD

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

Figura 4.1: Ejemplo de uso de BARD.

supone una herramienta útil para la extracción de información a partir de respuestas y la estructuración de la misma. Sin embargo, aunque indiques que se limite a darte la información pedida, en un amplio número de casos añade texto adicional como en este caso “¿Hay algo más que pueda hacer por ti.”. 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.

La interacción chatbot usuario de este prototipo se llevaba a cabo mediante una pequeña interfaz desarrollada con widgets de IPython. Esta versión del chatbot solicita al usuario información como nombre, edad, ciudad, etc., y al final muestra un mensaje de despedida con los datos recopilados. Cómo se puede ver en la figura



The image shows a dark-themed chatbot interface. At the top, it says "Bienvenido". Below that, there are four input sections, each with a label and a "Siguiete" button. The first section is "Nombre:" with a text box containing "Marta". The second section is "Edad:" with a slider control set to "20". The third section is "Ciudad:" with a text box containing "Calatayud". The fourth section is "Infancia:" with a text box containing "hola que tal me llamo marta".

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

4.6, la interacción con el usuario se hace mediante cajas en las que se escribe la información solicitada. Una vez rellanado se envía la información al chatbot para su análisis pulsando en el botón “Siguiete” que despliega un nuevo cuadro con la siguiente información solicitada.

Todo este desarrollo se llevo a cabo durante en 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 oficializo 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 primer prototipo paso a un segundo plano y se centro el trabajo de los siguientes meses en encontrar qué API usar para el desarrollo del trabajo. Se barajaron distintas posibilidades.

Por un lado se estudio 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 computo no asumible.

Por otro lado , se decidió estudiar distintos modelos de hugging face ¹ entre los que destaco gemma.

¹<https://huggingface.co/>

4.4. Prototipo usando GEMMA

Al comenzar a desarrollar 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 impedimenta 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 llegó 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 es “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 algunas funcionalidades y módulos hasta el final, como por ejemplo la interfaz o la generación de preguntas. Este parón se debió a un problema hardware. Al hacer instalarme Linux, no tenía apenas espacio en el disco duro por lo que la partición no fue 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 desarrollándolo en ese sistema operativo.

4.5. Desarrollo de la interfaz con RASA

Paralelamente a todo el trabajo descrito anteriormente, y durante el estudio de las diferentes alternativas de interfaces, APIs y modelos se estudio la alternativa de la api de rasa de la que ya se ha hablado en la sección 3.2.4.

Esta opción fue inicialmente barajada como alternativa a BARD o Gemma, sin embargo, su capacidad de generación de texto no cumplía los requisitos requeridos para este proyecto por lo que se descarto para tal uso.

Sin embargo, se encontró especialmente interesante debido a su fácil integración con Telegram para desarrollar chatbots. Como se puede ver en la figura 4.3 la interfaz generada de esta forma presenta numerosas ventajas.

En primer lugar, Telegram es una plataforma de mensajería ampliamente utilizada en todo el mundo, lo que garantiza un amplio alcance y accesibilidad para los usuarios. Esto significa que los usuarios pueden acceder al chatbot fácilmente desde su escritorio, pero también desde sus dispositivos móviles, lo que facilita su adopción y uso.

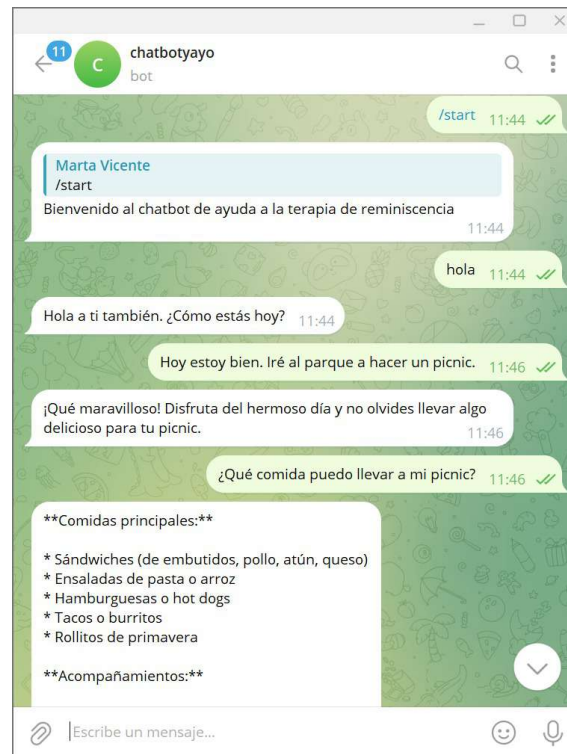


Figura 4.3: Ejemplo de uso de BARD.

Esta forma de generar la interfaz implica el desarrollo de manejadores que controlen el funcionamiento según el tipo de mensaje.

- Manejadores para comandos, por ejemplo, el comando start.
- Manejadores para mensajes de texto.
- Manejadores para imágenes.

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.

4.6. Gemini

A lo largo de los meses se estudiaron múltiples modelos y alternativas a BARD que no resultaban lo suficientemente eficaces. Por contrario, otras sí resultaron eficaces y pese a dar lugar a pequeños prototipos no pudieron seguir desarrollándose debido a los diversos problemas que se han contado a lo largo de este capítulo. Todo este estudio, implementación de modelos y desarrollo de prototipos fallidos conlleva horas de tiempo y esfuerzo. Todo esto deriva en meses centrados en análisis de alternativas en lugar del 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 Gemin. Como se ha comentado antes, esta alternativa se descarto en un primer momento por no estar disponible en España. Sin embargo este problema se pudo solventar fácilmente mediante el uso de VPN.

4.7. Análisis de las respuestas

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*.

4.7.1. Análisis de las respuestas y 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 en la figura ??.

¿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 está ocurriendo en todo momento.

Es por esto, que se han desarrollado una serie de funciones que se encargan de transformar la respuesta en un objeto *json* como se muestra en la figura 4.4. Este prototipo también añade información adicional que haya podido obtener de la respuesta y que no se pueda asociar a los campos predefinidos. Sin embargo, estos campos son útiles para llevar un control de qué información no ha podido ser obtenida y en tal caso, generar nuevas preguntas que nos permitan obtenerla.

```
>>>¿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'}
```

Figura 4.4: Análisis de respuestas

4.7.2. Generación de preguntas para obtener la información faltante

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 a los campos que no han sido rellenados tales como las que se muestran en la figura 4.5. Estas preguntas se van lanzando al usuario hasta rellenar los campos faltantes o hasta agotar el límite de preguntas asociadas a un mismo tema que se va a hacer al usuario.

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

Figura 4.5: Generación de preguntas

4.8. Interfaz e interacción con el usuario

Como se puede ver en la figura 3.7 estas bibliotecas permiten crear interfaces bastante usables y simples. Sin embargo, durante el estudio previo realizado apareció la API de Rasa (4.5) y su integración con Telegram.

La interfaz de Telegram es intuitiva y fácil de usar, lo que proporciona una experiencia de usuario más agradable y cómoda. Los usuarios pueden interactuar

con el chatbot de manera similar a como lo harían con otros contactos o grupos en la aplicación, lo que reduce la curva de aprendizaje y aumenta la aceptación del usuario.

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. Esto puede mejorar la efectividad del chatbot en la prestación de servicios de apoyo emocional.

4.8.1. Integración de Gemini y RASA

La decisión de desarrollar la interfaz del prototipo integrando RASA y GEMINI para tener una interfaz en Telegram tuvo como consecuencia la necesidad de llevar a cabo una refactorización.

4.8.1.1. Clase Pregunta

La 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 RASA 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 RASA, 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.

4.9. Extracción de información a partir de imágenes

La api de Gemini tiene muchas funcionalidades que permiten hacer 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 imágenes. 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.



Figura 4.6: Ejemplo de extracción de información a partir de imágenes

Capítulo 5

Herramienta conversacional de ayuda a la terapia a la reminiscencia

Este capítulo tiene como objetivo presentar la solución final desarrollada a partir de la problemática presentada 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 el objetivo de permitir la opción de estudiar los componentes de forma práctica, es decir, usando el prototipo, este capítulo comenzará por la puesta en marcha. 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 poderosos 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.

5.2. 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 (Software Development Kit). La API de Gemini está contenida en el paquete `google-generativeai` 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 tengas tu clave, puedes configurarla 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)
```

5.3. Arquitectura

5.3.1. Interfaz e interacción con el usuario

5.3.2. Almacenamiento y manejo de la información

5.3.3. Generación de historias de vida

Capítulo 6

Conclusiones y Trabajo Futuro

Conclusiones del trabajo y líneas de trabajo futuro.

Antes de la entrega de actas de cada convocatoria, en el plazo que se indica en el calendario de los trabajos de fin de grado, el estudiante entregará en el Campus Virtual la versión final de la memoria en PDF.

Conclusions and Future Work

Conclusions and future lines of work. This chapter contains the translation of Chapter 6.

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*. Proyecto Fin de Carrera, Facultad de Informática, Universidad Complutense de Madrid, 2021.
- 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.
- DEVLIN, J., CHANG, M.-W., LEE, K. y TOUTANOVA, K. Bert: Pre-training of deep bidirectional transformers for language understanding. En *NAACL*. 2019.
- 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.

- 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.
- MAGAZ, L. L. Grado en ingeniería informática. 2023. Curso 2022-2023.
- MIKOLOV, T., SUTSKEVER, I., K. CHEN, G. C., y DEAN, J. Distributed 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.
- ROTHMAN, D. *Transformers for Natural Language Processing*. Packt Publishing, 2nd edición, 2022.
- 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.
- SLOVIOVSKAYA, V. Transfer learning from transformers to fake news challenge stance detection (fnc-1) task. *arXiv preprint arXiv:1910.14353*, 2019.
- 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.
- 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.

