



**uhu.es**

**Escuela Técnica Superior de Ingeniería  
Universidad de Huelva**

Grado en Ingeniería Informática

**Trabajo Fin de Grado**

**Desarrollo de un Sistema Asistente de Música  
Basado en Aprendizaje Automático y Procesamiento  
del Lenguaje Natural con Modelos de Lenguaje de  
Gran Escala**

Autor: Daniel Linfon Ye Liu

Tutor: Jacinto Mata Vázquez

junio, 2025

*A mi abuelo,  
que siempre vivirá en mi corazón y en mi memoria.*

# Resumen

---

La música, presente en casi todos los aspectos de la vida cotidiana, ha visto transformando su acceso y consumo con el avance de las tecnologías digitales. Entre estos cambios, los asistentes musicales inteligentes han introducido una nueva forma de interactuar con plataformas de streaming y bibliotecas digitales, facilitando el uso mediante comandos en lenguaje natural. Sin embargo, muchos de estos sistemas siguen teniendo dificultades para comprender con precisión el contexto en el que se formulan las peticiones del usuario, lo que limita la calidad de la interacción.

Este trabajo se centra en el desarrollo de un asistente conversacional de música, combinando técnicas de procesamiento del lenguaje natural (NLP) con métodos de aprendizaje automático. El objetivo es mejorar la capacidad del sistema para interpretar instrucciones musicales emitidas en lenguaje natural. Para ello, se emplean modelos de lenguaje avanzados basados en arquitecturas Transformer, concretamente BERT, RoBERTa y T5, que han demostrado un notable rendimiento en tareas lingüísticas complejas.

La metodología se basa en el entrenamiento y evaluación de modelos orientados a la clasificación de texto. Estas técnicas se aplican a funcionalidades concretas, como la gestión de listas de reproducción (añadir, eliminar, mostrar o limpiar canciones) y la generación de recomendaciones personalizadas. Para ello, se construye un dataset específico con etiquetas asociadas a intenciones musicales, sobre el cual se entrena los modelos capaces de interpretar las entradas del usuario.

Los resultados experimentales reflejan una mejora significativa en la interpretación del lenguaje natural gracias al uso de modelos Transformer, en comparación con enfoques tradicionales. En definitiva, se demuestra la viabilidad de construir un asistente musical más preciso y adaptativo, capaz de ofrecer una experiencia de uso más coherente, natural y personalizada.

**Palabras clave:** Procesamiento del Lenguaje Natural, Aprendizaje Automático, Asistente Conversacional, Recomendación Musical, Transformers, BERT, RoBERTa, T5, Lenguaje Natural, Chatbot, Sistemas Inteligentes, Interacción Hombre-Máquina

# Abstract

---

Music, present in almost every aspect of daily life, has seen its access and consumption transformed with the advancement of digital technologies. Among these changes, intelligent music assistants have introduced a new way of interacting with streaming platforms and digital libraries, facilitating use through natural language commands. However, many of these systems still struggle to accurately understand the context in which user requests are formulated, which limits the quality of the interaction.

This work focuses on the development of a conversational music assistant, combining natural language processing (NLP) techniques with machine learning methods. The goal is to improve the system's ability to interpret musical instructions issued in natural language. To achieve this, advanced language models based on Transformer architectures, such as BERT, RoBERTa, and T5, are used, which have demonstrated remarkable performance in complex linguistic tasks.

The methodology is based on the training and evaluation of models aimed at both classification and text generation. These techniques are applied to specific functionalities, such as song search, playlist management (adding, removing, displaying, or clearing songs), and generating personalized recommendations. To achieve this, a specific dataset with labels associated with musical intents is built, on which a chatbot capable of maintaining a fluid conversation with the user is trained.

The experimental results reflect a significant improvement in natural language interpretation thanks to the use of Transformer models, compared to traditional approaches. Ultimately, the feasibility of building a more accurate and adaptive music assistant is demonstrated, capable of offering a more coherent, natural, and personalized user experience.

**Keywords:** Natural Language Processing, Machine Learning, Conversational Assistant, Music Recommendation, Transformers, BERT, RoBERTa, T5, Natural Language, Chatbot, Intelligent Systems, Human-Machine Interaction

# Agradecimientos

---

Quiero expresar mi más sincero agradecimiento a todas las personas que han hecho posible la realización de este Trabajo Fin de Grado.

En primer lugar, gracias a mis compañeros de clase por todos los momentos compartidos y por toda su ayuda a lo largo de la carrera. Agradecer también a mis amigos de la infancia, por los innumerables momentos vividos desde que era pequeño. Por estar ahí siempre que los necesite.

Mi gratitud también a los tutores y docentes que me guiaron con paciencia y dedicación, aportando sus conocimientos y consejos en cada etapa de mi vida universitaria.

Un agradecimiento especial a mis amigos del Erasmus en Stavanger, Noruega. Por acompañarme y animarme durante todo el desarrollo de este TFG y por hacer de esta experiencia, probablemente, la mejor que tendré en mi vida. Siempre os recordaré con muchísimo cariño. #SørmarkaFamily.

Por último, a mi familia, por haberme dado siempre todo su cariño y confianza incondicional. Por los valores que me han enseñado, que me han convertido en la persona que soy a día de hoy. Por todo lo que han luchado para que yo haya vivido una infancia tan bonita y pueda disfrutar de la vida. Siempre estaré eternamente agradecido con vosotros.

A todos vosotros, mil gracias.

*Daniel Linfon*

Huelva, 2025

# Índice general

---

<b>Resumen</b>	<b>II</b>
<b>Abstract</b>	<b>III</b>
<b>Agradecimientos</b>	<b>IV</b>
<b>Índice de Figuras</b>	<b>VII</b>
<b>Índice de Tablas</b>	<b>VIII</b>
<b>1. Introducción</b>	<b>1</b>
1.1. Motivación . . . . .	1
1.2. Objetivos . . . . .	2
1.3. Competencias . . . . .	2
1.4. Estructura de la Memoria . . . . .	3
<b>2. Marco Teórico</b>	<b>4</b>
2.1. Aprendizaje Automático . . . . .	4
2.2. Clasificación de texto por intención . . . . .	5
2.3. Redes Neuronales . . . . .	6
2.3.1. Aprendizaje profundo . . . . .	7
2.3.2. Epochs y tamaño de lote . . . . .	7
2.3.3. Tasa de aprendizaje . . . . .	8
2.3.4. Función de pérdida . . . . .	8
2.3.5. Overfitting y técnicas de regularización . . . . .	10
2.4. Procesamiento del Lenguaje Natural . . . . .	11
2.4.1. Representación del texto . . . . .	11
2.5. Transformers . . . . .	13
2.5.1. Modelos de lenguaje basados en Transformers . . . . .	15
2.6. Aprendizaje por Transferencia . . . . .	16
2.6.1. Fine-tuning y optimización de hiperparámetros . . . . .	16
2.7. Ensemble de Modelos . . . . .	17
2.8. Medidas de Evaluación . . . . .	17
2.8.1. Accuracy, Precision, Recall y F1-Score . . . . .	18
2.8.2. Trabajos Relacionados . . . . .	19
2.9. Tecnologías y Recursos Utilizados . . . . .	19
<b>3. Metodología, Experimentación y Resultados</b>	<b>22</b>
3.1. Descripción General de la Metodología . . . . .	22
3.2. Preparación de los Datos . . . . .	23
3.2.1. Descripción de los Datos . . . . .	23
3.2.2. Limpieza y Normalización . . . . .	24
3.2.3. Tokenización y Codificación . . . . .	24
3.3. Selección de los Modelos . . . . .	25
3.4. Ajuste de Hiperparámetros . . . . .	26
3.4.1. Métodos de Búsqueda . . . . .	26

3.4.2. Hiperparámetros Evaluados . . . . .	27
3.4.3. Tamaño del Conjunto de Datos Utilizado . . . . .	27
3.4.4. Criterios de Selección . . . . .	27
3.5. Entrenamiento de los Modelos . . . . .	28
3.5.1. Configuración de Entrenamiento . . . . .	28
3.5.2. Monitorización del Entrenamiento . . . . .	28
3.6. Evaluación y Análisis de Resultados . . . . .	31
3.7. Análisis de Errores . . . . .	34
3.7.1. Errores más frecuentes . . . . .	34
3.8. Implementación de Chatbot . . . . .	36
3.8.1. Base de Datos . . . . .	36
3.8.2. Arquitectura del Sistema . . . . .	37
3.8.3. Integración del Modelo . . . . .	38
3.8.4. Funcionalidades del sistema . . . . .	38
3.8.5. Resultado Final . . . . .	39
<b>4. Conclusiones y Trabajo Futuro</b>	<b>46</b>
4.1. Conclusiones . . . . .	46
4.2. Trabajo Futuro . . . . .	47
4.3. Planificación Temporal del Trabajo Realizado . . . . .	48
<b>Referencias</b>	<b>50</b>
<b>A. Anexos</b>	<b>52</b>
A.1. Repositorio de código utilizado . . . . .	52

# Índice de Figuras

---

2.1. Esquema general del procesamiento de texto en el asistente conversacional . . . . .	5
2.2. Diagrama de la neurona de una red neuronal . . . . .	6
2.3. Cálculo de la función de pérdida utilizando <i>softmax</i> y <i>cross-entropy</i> . . . . .	9
2.4. Aplicación de Early Stopping a un problema de overfitting . . . . .	10
2.5. Espacio vectorial formado por one-hot encoding vectors . . . . .	12
2.6. Arquitectura CBOW . . . . .	13
2.7. Arquitectura Skip-Gram . . . . .	13
2.8. Procedimiento general de generación de word embeddings . . . . .	14
2.9. Arquitectura de un Transformer . . . . .	15
2.10. Proceso de aprendizaje por transferencia . . . . .	16
2.11. Funcionamiento de un ensemble de modelos . . . . .	17
2.12. Tecnologías y recursos utilizados en el desarrollo del trabajo . . . . .	21
3.1. Diagrama general del flujo de trabajo seguido en el proyecto. . . . .	23
3.2. Distribución de ejemplos por intención en los subconjuntos de entrenamiento, validación y test. . . . .	24
3.3. Curvas de pérdida por época para BERT (bert-base-uncased). . . . .	29
3.4. Curvas de pérdida por época para RoBERTa (roberta-base). . . . .	30
3.5. Curvas de pérdida por época para T5 (t5-small). . . . .	30
3.6. Matriz de confusión en test para BERT-base. . . . .	33
3.7. Matriz de confusión en test para RoBERTa-base. . . . .	33
3.8. Matriz de confusión en test para T5-small. . . . .	34
3.9. Arquitectura del Chatbot . . . . .	37
3.10. Captura de pantalla del chatbot al iniciar el programa. . . . .	39
3.11. Captura de pantalla del chatbot al añadir una canción indicando el título y artista. . . . .	40
3.12. Captura de pantalla del chatbot al añadir una canción indicando únicamente el título. . . . .	41
3.13. Captura de pantalla del chatbot al solicitar ver la playlist del usuario . . . .	42
3.14. Captura de pantalla del chatbot al solicitar eliminar una canción de la playlist	43
3.15. Captura de pantalla del chatbot al solicitar vaciar la playlist . . . . .	44

# Índice de Tablas

---

2.1. Matriz de confusión binaria utilizada para definir las métricas de evaluación.	18
2.2. Matriz de confusión multiclas entre las intenciones predichas y las reales.	18
3.1. Espacio de búsqueda definido para los hiperparámetros evaluados con Optuna.	27
3.2. Hiperparámetros óptimos encontrados por Optuna para cada modelo.	28
3.3. Resultados de evaluación sobre el conjunto de test	31
3.4. Resultados por etiqueta para los modelos evaluados sobre el conjunto de test.	31
3.5. Estructura de la tabla <code>user_songs</code>	36
4.1. Planificación temporal del trabajo realizado.	49

## CAPÍTULO 1

# Introducción

---

Este primer capítulo sirve de introducción y brinda una base general del proyecto. En primer lugar, se describe la motivación por la cual se ha decidido llevar a cabo este trabajo, los objetivos propuestos y las competencias que se pretenden adquirir. Finalmente, se detalla la estructura del documento para guiar al lector a lo largo de los distintos capítulos.

### 1.1. Motivación

La forma en que las personas acceden y gestionan su música ha cambiado radicalmente con la aparición de asistentes virtuales y servicios de streaming. Actualmente, los usuarios pueden interactuar con sus bibliotecas musicales mediante comandos de voz o texto, facilitando tareas como buscar canciones, gestionar listas de reproducción o recibir recomendaciones personalizadas.

Sin embargo, muchos de estos sistemas presentan limitaciones importantes en la interpretación del lenguaje natural, lo que genera interacciones imprecisas o frustrantes. Estas carencias evidencian la necesidad de explorar nuevas técnicas que permitan una comprensión más profunda y contextual de las intenciones del usuario.

Este trabajo surge con el propósito de abordar esa necesidad mediante el desarrollo de un asistente conversacional musical que utilice modelos avanzados de procesamiento del lenguaje natural (PLN) basados en Transformers. La integración de estos modelos busca ofrecer una interacción más natural y efectiva, superando las limitaciones de los asistentes tradicionales.

La idea de desarrollar este asistente conversacional surgió a raíz de un trabajo que realicé durante mi erasmus en Stavanger, Noruega. Fue esa experiencia la que me motivó a continuar explorando el potencial de los modelos de lenguaje y su aplicación en asistentes conversacionales, llevándome a elegir este tema para mi Trabajo Fin de Grado.

### 1.2. Objetivos

El objetivo principal de este proyecto es diseñar, implementar y evaluar un asistente conversacional de música capaz de comprender comandos en lenguaje natural y gestionar listas de reproducción de forma contextual.

Para alcanzar este propósito, se han definido los siguientes objetivos específicos:

- Analizar el estado del arte en el uso de modelos de lenguaje para asistentes conversacionales.
- Recopilar y preprocesar un conjunto de datos etiquetados con intenciones musicales (añadir, eliminar, ver, limpiar).
- Optimizar los parámetros de los modelos de lenguaje utilizando Optuna para encontrar la mejor configuración y mejorar el rendimiento.
- Implementar y entrenar modelos de lenguaje para la comprensión de intenciones.
- Evaluar el rendimiento del sistema en tareas de clasificación y generación de texto, utilizando métricas como precisión y F1-score.
- Comparar los resultados obtenidos entre distintos modelos y enfoques.
- Diseñar un flujo conversacional que permita al usuario interactuar con el asistente mediante comandos en lenguaje natural.

### 1.3. Competencias

La principal competencia adquirida con la realización de este trabajo ha sido:

**CE7-C - Capacidad para conocer y desarrollar técnicas de aprendizaje computacional y diseñar e implementar aplicaciones y sistemas que la utilicen.**

Además, se han desarrollado otras competencias clave del plan de estudios:

- **Capacidad para conocer y desarrollar técnicas de aprendizaje computacional.** A lo largo del proyecto se ha trabajado con modelos de lenguaje avanzados y técnicas de clasificación de texto.
- **Capacidad para implementar aplicaciones y sistemas que utilicen inteligencia artificial.** Se ha implementado un sistema funcional que permite la interacción conversacional con el usuario para la gestión musical.
- **Capacidad para la extracción automática de información.** El asistente identifica y extrae intenciones específicas desde entradas de lenguaje natural.

## **1. Introducción**

---

- **Capacidad para adquirir conocimiento a partir de conjuntos de datos.** Se han utilizado datasets con múltiples ejemplos de interacciones para entrenar y ajustar los modelos de lenguaje.

### **1.4. Estructura de la Memoria**

El resto de esta memoria se organiza de la siguiente manera:

- En el [Capítulo 2](#), Marco Teórico, se exponen los conceptos fundamentales relacionados el procesamiento del lenguaje natural y los modelos de lenguaje Transformer utilizados en este trabajo.
- En el [Capítulo 3](#) se describe la metodología empleada para el desarrollo del sistema, incluyendo la recolección de datos, el entrenamiento y evaluación de los modelos y el diseño del flujo conversacional.
- En el [Capítulo 4](#) se presentan las conclusiones del trabajo, junto con posibles líneas futuras de mejora y ampliación del sistema. Además, se incluye una planificación temporal con las horas estimadas dedicadas a cada fase del trabajo.
- Finalmente, en el [Apéndice A](#) se incluye material complementario como el repositorio con el código desarrollado y el material utilizado para la implementación del sistema.

## CAPÍTULO 2

# Marco Teórico

---

En este capítulo se introducen los conceptos teóricos necesarios para entender en qué campos nos hemos centrado a la hora de enfocar este proyecto.

### 2.1. Aprendizaje Automático

El aprendizaje automático, también conocido como *Machine Learning*, es una rama de la inteligencia artificial que se centra en el diseño de algoritmos capaces de aprender directamente a partir de datos. Estos algoritmos detectan patrones y relaciones dentro de la información que reciben, lo que les permite realizar tareas concretas y mejorar su rendimiento a medida que acumulan experiencia.

En función de cómo se estructuren los datos y el objetivo del aprendizaje, los algoritmos suelen clasificarse en tres categorías: [1]

- **Aprendizaje Supervisado.** El algoritmo se entrena utilizando un conjunto de datos etiquetado, es decir, con entradas de las que se conoce la salida esperada. A partir de estos datos etiquetados, el sistema aprende a predecir nuevas salidas cuando se le presentan entradas similares. Entre las aplicaciones más comunes de estos algoritmos se encuentra el problema de clasificación.
- **Aprendizaje no Supervisado.** A diferencia del caso anterior, aquí no se dispone de etiquetas. El algoritmo trabaja con datos sin clasificar y busca de forma autónoma estructuras o agrupaciones ocultas en ellos. Esto es útil, por ejemplo, en tareas de segmentación o agrupación de usuarios según su comportamiento, sin necesidad de conocer de antemano a qué grupo pertenecen.
- **Aprendizaje por Refuerzo.** En este caso, el modelo aprende mediante prueba y error, interactuando con un entorno y recibiendo recompensas o penalizaciones según las decisiones que toma. Su objetivo es aprender a tomar las mejores decisiones posibles para maximizar una señal de recompensa acumulada. Este tipo de aprendizaje se ha aplicado con éxito en campos como la robótica, videojuegos y sistemas de recomendación.

En este proyecto se ha optado por el uso de **aprendizaje supervisado**, ya que se dispone de un conjunto de datos etiquetado en el que cada ejemplo de entrada es una

## 2. Marco Teórico

---

instrucción en lenguaje natural y está asociado a una etiqueta que representa la intención del usuario (añadir o eliminar canción, ver y vaciar playlist). Esto ha permitido entrenar modelos de clasificación basados en arquitecturas Transformer, incluyendo modelos de gran escala (LLMs), capaces de interpretar comandos musicales con mayor precisión.

### 2.2. Clasificación de texto por intención

La clasificación de texto es una tarea esencial dentro del aprendizaje automático supervisado. Consiste en asignar una categoría o etiqueta a una secuencia de palabras en función de su contenido. En el caso de nuestro chatbot, esta técnica se aplica para detectar la intención del usuario a partir de una entrada en lenguaje natural.

Entre las acciones posibles se incluyen añadir una canción, eliminarla, visualizar la lista actual o vaciarla por completo. Se trata, por tanto, de un problema de clasificación **multiclas**, donde cada mensaje debe ser etiquetado con una única categoría entre un conjunto definido de intenciones.

Para procesar los textos, se utilizan modelos de lenguaje preentrenados basados en Transformers. Estos modelos convierten las secuencias de texto en representaciones numéricas que capturan su significado dentro del contexto, lo que permite analizarlas con mayor precisión. A partir de estas representaciones, se entrena un clasificador que determina la intención más probable asociada al mensaje. El uso de este tipo de modelos ha demostrado ser especialmente eficaz en tareas de comprensión del lenguaje natural, ya que permite capturar relaciones semánticas complejas entre las palabras [2].

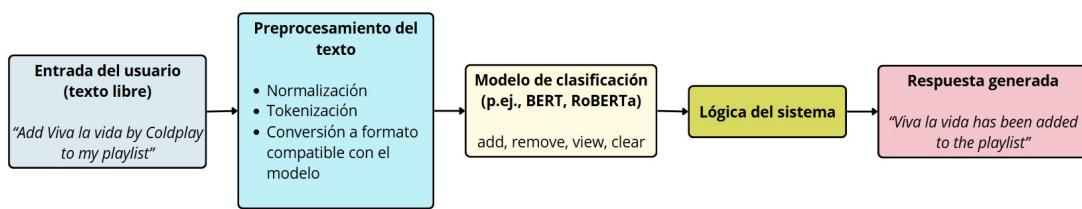


Figura 2.1: Esquema general del procesamiento de texto en el asistente conversacional

Como se muestra en la Figura 2.1, nuestro sistema recibe una entrada en lenguaje natural que es procesada por un modelo de clasificación. Este modelo identifica la intención del usuario y activa la acción correspondiente sobre la lista de reproducción. Esta estructura permite una interacción conversacional fluida y adaptada al contexto, mejorando la experiencia del usuario en la gestión de su música.

## 2.3. Redes Neuronales

Las redes neuronales artificiales se han consolidado como una de las herramientas más potentes en el ámbito de la inteligencia artificial, especialmente por su capacidad para resolver problemas complejos en áreas como la visión por computador o el procesamiento del lenguaje natural. Su diseño se inspira en el funcionamiento del cerebro humano, donde las neuronas se conectan entre sí para transmitir información. De forma similar, una red neuronal artificial está compuesta por capas de nodos o neuronas artificiales que convierten una señal de entrada en una salida mediante operaciones matemáticas [3].

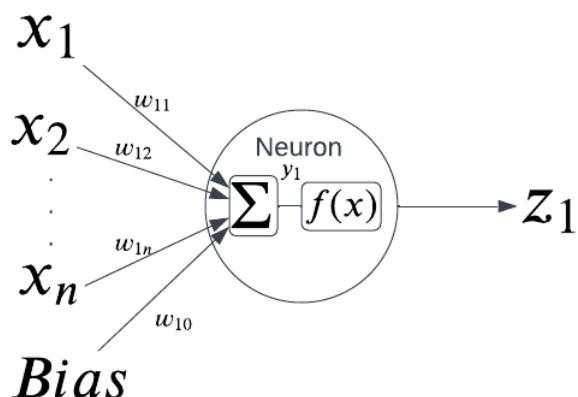


Figura 2.2: Diagrama de la neurona de una red neuronal

Cada neurona artificial recibe una serie de entradas y las pondera mediante unos pesos ajustables. Luego, añade un término conocido como *Bias* que permite desplazar la función de activación y aplica una función de activación que genera la salida correspondiente. Esta salida se transmite a la siguiente capa, lo que permite que la información fluya a lo largo de la red y se vaya transformando en cada paso [4].

Durante el entrenamiento, la red ajusta tanto los pesos como los biases para minimizar el error de predicción. Este ajuste se realiza mediante algoritmos como la retropropagación del error (*backpropagation*), que calcula el gradiente de la función de pérdida respecto a cada parámetro mediante la regla de la cadena. Gracias a este mecanismo, el modelo puede aprender de sus errores y mejorar su rendimiento general [5].

Una de las ventajas de este tipo de redes es su estructura jerárquica, que permite aprender representaciones progresivamente más abstractas a partir de los datos. Esto las hace especialmente útiles en tareas como la clasificación, la predicción o la generación de contenido.

## 2. Marco Teórico

---

Cuando estas redes se construyen con múltiples capas ocultas, se da paso a un paradigma más avanzado conocido como aprendizaje profundo, que ha ampliado de forma notable las capacidades de los sistemas de inteligencia artificial en los últimos años.

### 2.3.1. Aprendizaje profundo

El aprendizaje profundo (*Deep Learning*) es una subdisciplina del aprendizaje automático que ha transformado el tratamiento de datos complejos como imágenes, audio o texto. A diferencia de los enfoques clásicos, que requieren una extracción manual de características, el aprendizaje profundo permite que los modelos descubran automáticamente las representaciones más útiles a partir de los datos.

Se basa en redes neuronales artificiales con más de tres capas ocultas, conocidas como redes profundas. Su capacidad para trabajar con representaciones jerárquicas les permite resolver tareas que las redes superficiales (aquellas con una o dos capas ocultas) no pueden abordar con la misma eficacia [6].

En este proyecto, el aprendizaje profundo constituye la base del sistema conversacional, permitiendo interpretar comandos musicales expresados en lenguaje natural de forma precisa.

Antes de entrenar un modelo basado en aprendizaje profundo, es necesario definir una serie de parámetros que influyen directamente en su rendimiento y en la calidad de los resultados obtenidos. Entre estos parámetros destacan el número de épocas, el tamaño del lote, la tasa de aprendizaje o la función de pérdida, entre otros.

### 2.3.2. Epochs y tamaño de lote

Durante el entrenamiento de un modelo de aprendizaje automático, los datos no se procesan todos a la vez, sino que se organizan en partes más pequeñas para facilitar el ajuste de los parámetros del modelo. En este contexto, se define **epoch** como una pasada completa por todo el conjunto de entrenamiento. El número total de epochs determina cuántas veces se reutilizan los mismos datos para seguir optimizando el modelo [7].

Sin embargo, procesar todos los datos de una sola vez en cada iteración puede resultar ineficiente o inviable, sobre todo si el conjunto de datos es muy extenso. Por ello, se recurre a dividir el conjunto en subconjuntos más pequeños llamados **minibatches** o lotes, que contienen una cierta cantidad de muestras (*batch size*). Cada uno de estos lotes se procesa por separado, y tras cada uno se actualizan los parámetros del modelo en función del error cometido.

En este proyecto se ha optado por entrenar los modelos durante 10 epochs, ya que en las primeras pruebas se observó que este valor permitía al modelo alcanzar un buen rendimiento. En cuanto al batch size, se han probado diferentes valores (8, 16 y 32) y se ha ajustado dinámicamente durante la optimización.

### 2.3.3. Tasa de aprendizaje

Como habíamos mencionado anteriormente, durante el entrenamiento de una red neuronal se actualizan los parámetros del modelo (como los pesos) iterativamente en función del error cometido. La tasa de aprendizaje (*learning rate*) es uno de los hiperparámetros más relevantes en este proceso, ya que se encarga de determinar el tamaño del paso que da el optimizador en cada actualización, es decir, determina con qué rapidez el modelo se ajusta a los datos. Si la tasa de aprendizaje es muy alta, el modelo puede avanzar de forma inestable, sin llegar a aprender correctamente y divergir. Por el contrario, si es demasiado baja, los avances serán tan pequeños que el proceso de aprendizaje se volverá muy lento y puede que el modelo no consiga mejorar lo suficiente [8].

### 2.3.4. Función de pérdida

La función de pérdida se encarga de medir la diferencia entre las predicciones realizadas por el modelo y las etiquetas reales del conjunto de datos. Cuanto mayor es el error, mayor será el valor de esta función, y viceversa. Esta señal es la que permite al modelo ir ajustando sus parámetros poco a poco para mejorar [9].

Para nuestro caso, al tratarse de un problema de clasificación multiclase, se ha utilizado una función de pérdida llamada entropía cruzada (*cross-entropy*). Esta función no solo comprueba si el modelo acierta o se equivoca, sino que también tiene en cuenta cuánta seguridad tenía el modelo al tomar esa decisión. Si el modelo falla y lo hace con mucha confianza, la penalización será mayor [10].

Modelos como *BERT* y *RoBERTa* incluyen de forma nativa una capa **softmax** para convertir las salidas (*logits*) en probabilidades, lo que permite aplicar esta función directamente. Para *T5*, aunque su arquitectura es generativa, también puede emplearse *cross-entropy* al comparar la salida generada con la etiqueta esperada.

## 2. Marco Teórico

---

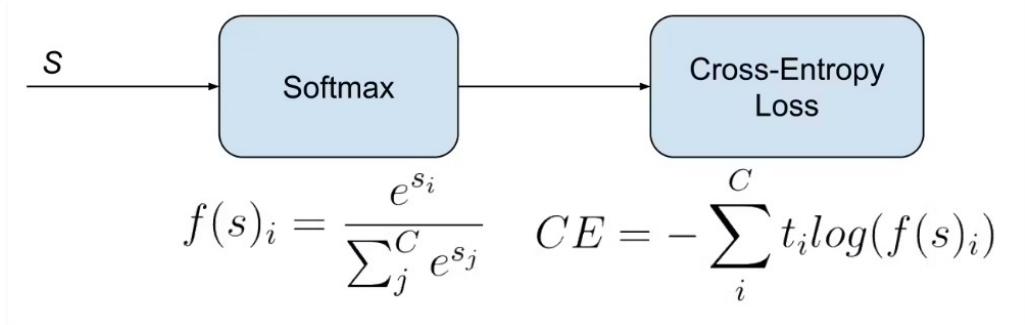


Figura 2.3: Cálculo de la función de pérdida utilizando *softmax* y *cross-entropy*.

Por ejemplo, supongamos que el agente recibe el mensaje “*I want to see my playlist*”, y la etiqueta correcta es `view`. Si tenemos cuatro clases posibles (`add`, `remove`, `view`, `clear`), esta clase correspondería al índice 3. El modelo predice las siguientes probabilidades:

$$p = [0,1, 0,2, \mathbf{0,6}, 0,2] \quad \text{y la etiqueta real es } t = [0, 0, \mathbf{1}, 0]$$

Estas probabilidades son el resultado de aplicar la función *softmax* a los logits generados por el modelo. Esta transformación convierte los valores sin normalizar en una distribución de probabilidad sobre las clases posibles, permitiendo aplicar la entropía cruzada de forma adecuada:

$$CE(t, p) = -\sum_{i=1}^C t_i \cdot \log(p_i) = -\log(0,6) \approx 0,5108 \quad (2.1)$$

Si el modelo hubiese asignado menor probabilidad a la clase correcta, por ejemplo:

$$p' = [0,2, 0,2, \mathbf{0,3}, 0,3]$$

La pérdida sería mayor:

$$CE(t, p') = -\log(0,3) \approx 1,204$$

Este aumento refleja que el modelo estaba menos seguro al predecir la clase correcta. La entropía cruzada no solo penaliza los errores, sino que también recompensa la confianza bien colocada. Si el modelo acierta pero con poca seguridad, la pérdida sigue siendo alta. Por eso, esta función no solo busca que el modelo acierte, sino que también aprenda a estar seguro cuando da la respuesta correcta.

### 2.3.5. Overfitting y técnicas de regularización

El sobreajuste (*overfitting*) ocurre cuando el modelo aprende excesivamente bien los datos del conjunto de entrenamiento y pierde la capacidad de generalizar a datos nuevos. El modelo memoriza ejemplos concretos en lugar de aprender patrones, lo que resulta en un buen rendimiento en el entrenamiento, pero un mal desempeño cuando se enfrenta a ejemplos no vistos. Para prevenir este problema, se han utilizado distintas técnicas de regularización.

Por un lado, se ha empleado el **dropout**, que consiste en desactivar aleatoriamente ciertas neuronas durante el entrenamiento. Con esto se consigue que el modelo no dependa en exceso de rutas concretas dentro de la red, lo que mejora su capacidad para generalizar. En este proyecto se ha fijado una probabilidad de desactivación del 30% ( $\text{dropout} = 0.3$ ).

Por otro lado, se ha empleado la técnica de **weight decay**, que actúa como un mecanismo de regularización L2. Su objetivo es evitar que los pesos del modelo crezcan demasiado durante el entrenamiento. Para lograrlo, se añade un pequeño término extra a la función de pérdida que penaliza los pesos grandes. De este modo, el modelo se ve presionado a mantener sus parámetros en valores más pequeños, lo que favorece que aprenda patrones más generales y útiles, en lugar de memorizar detalles concretos del conjunto de entrenamiento.

Por último, se ha incorporado el **early stopping**, una técnica que permite detener el entrenamiento cuando el modelo deja de mejorar su rendimiento en el conjunto de validación.

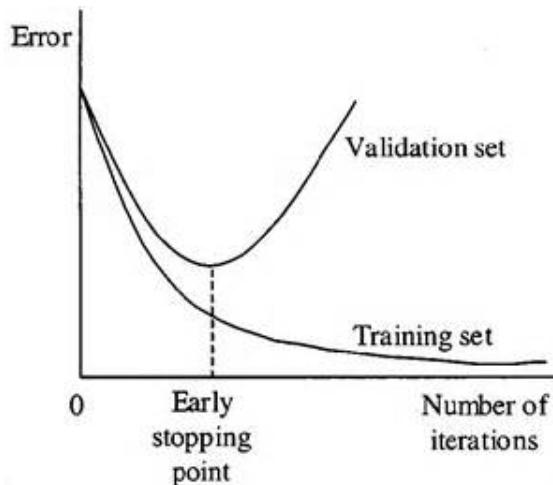


Figura 2.4: Aplicación de Early Stopping a un problema de overfitting

Como se observa en la Figura 2.4, al inicio del proceso el error disminuye tanto en el conjunto de entrenamiento como en el de validación. Tras cierto punto, el modelo empieza a ajustarse demasiado a los datos de entrenamiento, lo que provoca un aumento del error en validación. Es en este momento cuando el early stopping detecta el overfitting y detiene el entrenamiento justo antes de que el modelo empiece a degradarse, evitando así que

## 2. Marco Teórico

---

pierda capacidad de generalización. En este proyecto, se ha utilizado esta técnica con una paciencia de tres épocas.

### 2.4. Procesamiento del Lenguaje Natural

El Procesamiento del Lenguaje Natural (PLN) es una rama de la IA centrada en el diseño de sistemas capaces de comprender e interpretar el lenguaje humano. Su objetivo es permitir comunicarnos con las computadoras utilizando el lenguaje natural para facilitar la interacción hombre-máquina y para modelar tareas lingüísticas complejas, como la traducción automática, los resúmenes de texto o la recuperación de información [11].

Desde la perspectiva del PLN, el lenguaje humano está compuesto por múltiples niveles: fonológico, morfológico, sintáctico, semántico y pragmático. Estos niveles se combinan para expresar intenciones, contexto y significado. Para que un sistema pueda interpretar correctamente una frase, debe ser capaz de analizar estos diferentes planos del lenguaje. Esta complejidad hace que el procesamiento del lenguaje natural sea una tarea desafiante, especialmente en entornos conversacionales donde las expresiones pueden ser ambiguas, incompletas, informales, etc.

En el contexto de este trabajo, el PLN permite al sistema interpretar instrucciones expresadas de forma natural por el usuario y transformarlas en entradas útiles para los modelos de aprendizaje automático. Este proceso se apoya en modelos de lenguaje modernos basados en arquitecturas Transformer, capaces de capturar tanto el contexto como la estructura del lenguaje.

#### 2.4.1. Representación del texto

Una de las cuestiones clave a la hora de aplicar redes neuronales al lenguaje natural es cómo transformar el texto en datos numéricos que resulten útiles para los modelos. Para realizar esta conversión, la unidad fundamental es el **token**, que puede ser una palabra, un carácter o incluso un fragmento de palabra.

Para que nuestra red pueda procesar los tokens, si simplemente se asignara un número a cada uno, el modelo podría deducir relaciones inexistentes entre ellos basándose solo en la secuencia numérica. Por ello, se introdujo el **one-hot encoding**, donde cada token se representa como un vector de longitud igual al tamaño del vocabulario, donde solo una posición toma el valor 1 (indicando el token correspondiente) y el resto son ceros (Ver Figura 2.5). [12].

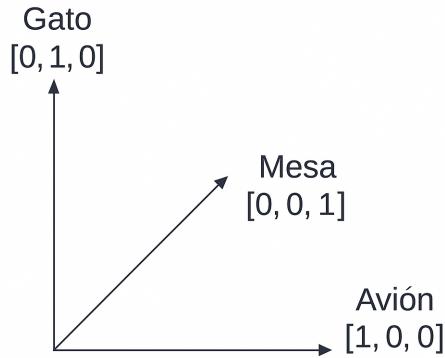


Figura 2.5: Espacio vectorial formado por one-hot encoding vectors

Aunque esto soluciona el problema de las relaciones numéricas artificiales, genera vectores muy grandes y poco informativos. La solución a estas limitaciones llegó con el desarrollo de los **Word Embeddings**.

Un Word Embedding es una representación numérica de palabras donde cada término se asocia a un vector de dimensiones reducidas, pero que capta relaciones de similitud y contexto. Así, palabras que aparecen en contextos parecidos tienden a tener vectores cercanos en el espacio, reflejando su afinidad semántica. [13]. Entre los métodos más conocidos para generar estos vectores destacan Word2Vec y GloVe.

**Word2Vec**, desarrollado por Google en 2013, convierte palabras en vectores de manera que las palabras que aparecen en contextos similares acaban teniendo vectores parecidos [14]. Utiliza dos tipos de arquitecturas diferentes para aprender estos vectores:

- **Skip-Gram:** Intenta predecir las palabras del contexto más próximas a una palabra dada (Figura 2.6).
- **CBOW (Continuous Bag of Words):** Hace lo contrario, toma varias palabras del contexto y trata de predecir la palabra central (Figura 2.7).

**GloVe**, propuesto por el equipo de Stanford, también representa palabras como vectores, pero en vez de fijarse solo en pequeñas ventanas de texto, analiza todo el corpus y cuenta con qué frecuencia aparecen juntas las palabras en cualquier parte [15].

En la Figura 2.8 se puede observar el procedimiento general de generación de word embeddings a partir de una representación inicial one-hot encoding. Cada palabra, representada inicialmente como un vector disperso, atraviesa una capa de embedding que transforma esa información en un vector denso de dimensiones mucho menores, pero cargado de significado semántico.

## 2. Marco Teórico

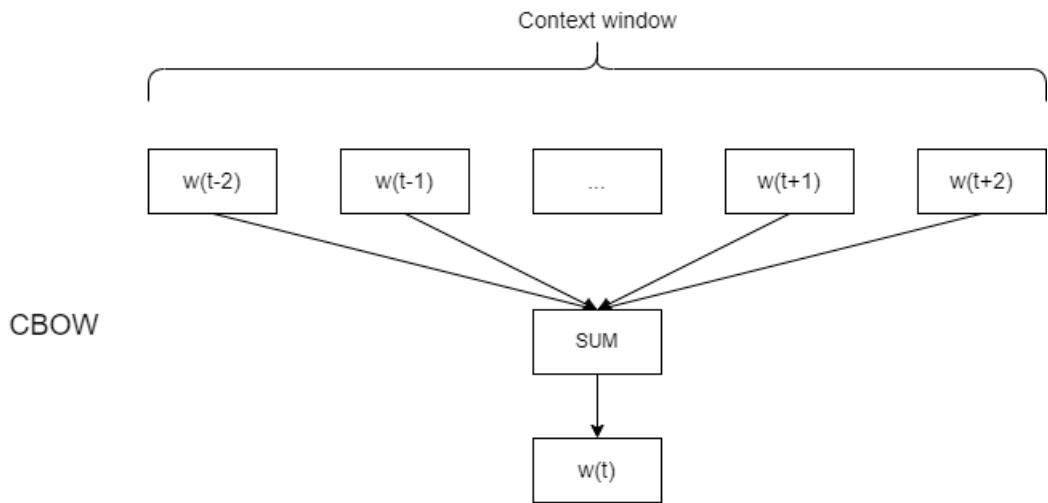


Figura 2.6: Arquitectura CBOW

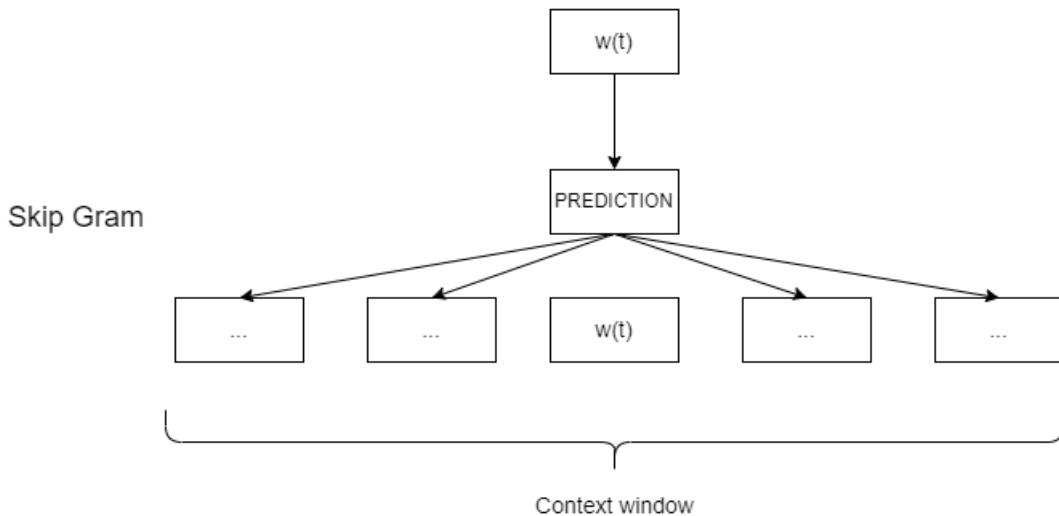


Figura 2.7: Arquitectura Skip-Gram

## 2.5. Transformers

Una vez que los textos han sido tokenizados y representados mediante embeddings, el siguiente paso es hacer que un modelo sea capaz de interpretar el significado de esos vectores. Para ello, el modelo que ha revolucionado esta etapa en los últimos años es el **Transformer**, introducido por Vaswani et al. en 2017 en su trabajo “*Attention is All You Need*” [16].

A diferencia de los embeddings estáticos, que asignan a cada palabra un único vector fijo sin considerar el contexto en el que aparece, los Transformers sí lo hacen gracias a una capa clave dentro de su arquitectura: el mecanismo de **auto-atención** (*self-attention*). Esto

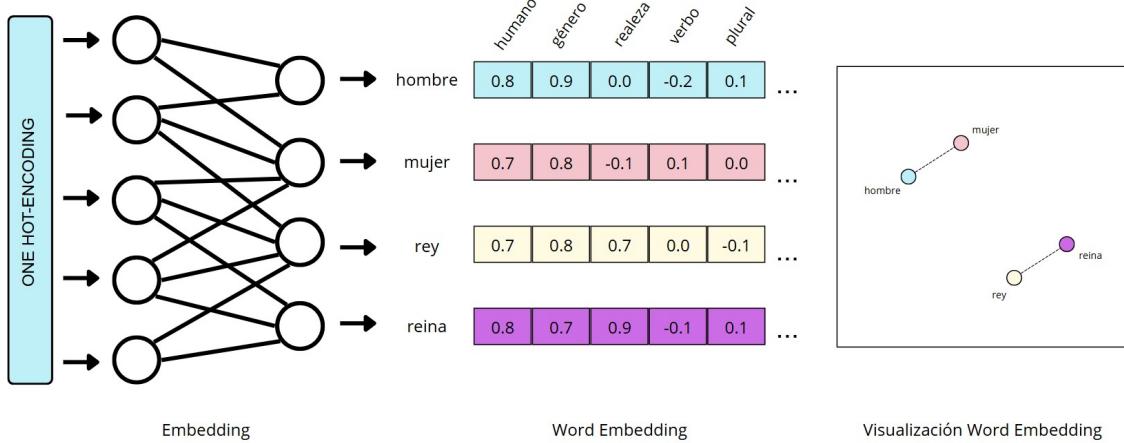


Figura 2.8: Procedimiento general de generación de word embeddings

permite al modelo identificar y ponderar la importancia relativa de cada palabra respecto a las demás, lo que lo convierte en una herramienta muy útil para tareas de clasificación textual [17].

Los Transformers son modelos de tipo *sequence-to-sequence* (S2S), es decir, están diseñados para transformar una secuencia de entrada en otra secuencia de salida. Su arquitectura está formada por dos componentes principales: un **codificador** (encoder), que procesa y representa la entrada, y un **decodificador** (decoder), que genera la salida final a partir de dicha representación. Esta estructura se ilustra en la Figura 2.9.

Sin embargo, distintos modelos han adoptado variantes de esta arquitectura dependiendo de la tarea para la que fueron diseñados:

- **Encoder-only:** emplean únicamente la parte del codificador. Modelos como *BERT* y *RoBERTa*.
- **Decoder-only:** basado únicamente en el decodificador. Modelos como *GPT* y *LLaMA*.
- **Encoder-decoder:** utilizan tanto encoder como decoder. Modelos como *T5*, *BART* o *Mistral*.

En este proyecto se han utilizado varios modelos basados en Transformers para clasificar las intenciones del usuario. Gracias al mecanismo de atención, estos modelos pueden detectar qué palabras son más relevantes en cada frase para deducir si el usuario quiere, por ejemplo, añadir o eliminar una canción. Esto convierte a los Transformers en una herramienta muy útil para tareas de clasificación textual.

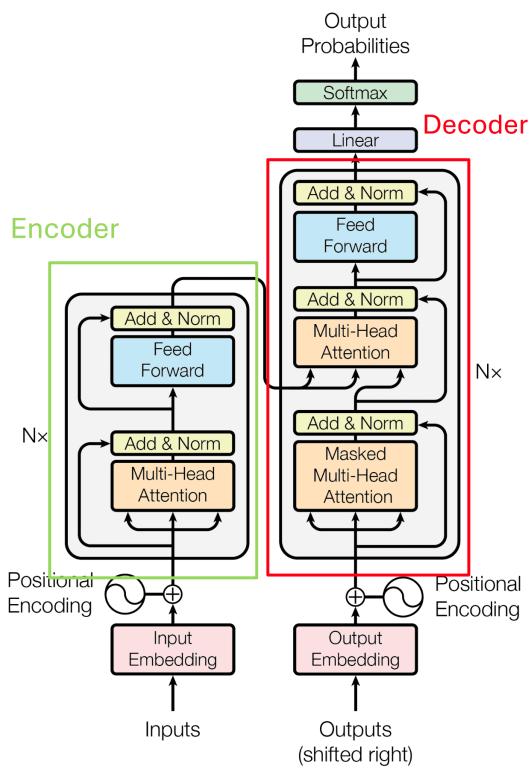


Figura 2.9: Arquitectura de un Transformer

### 2.5.1. Modelos de lenguaje basados en Transformers

En este trabajo se han empleado tres modelos basados en la arquitectura Transformer: *BERT* [18], *RoBERTa* [19] y *T5* [20].

Tanto *BERT* (Bidirectional Encoder Representations from Transformers) como *RoBERTa*, que es una versión optimizada de BERT, son modelos que utilizan únicamente la parte de codificador (*encoder-only*) del Transformer. Destacan por procesar la entrada de forma bidireccional, capturando el contexto completo de cada palabra dentro de la frase. Esta capacidad los convierte en modelos adecuados para tareas de clasificación como la que se plantea en este proyecto.

Por otro lado, *T5* (Text-to-Text Transfer Transformer) utiliza el codificador y decodificador (*encoder-decoder*), lo que le permite abordar diversas tareas de lenguaje natural. Por ejemplo, al ser capaz de manejar la generación de texto, podría ser bastante útil por si en un futuro se decide ampliar las funcionalidades del sistema. Sin embargo, en este proyecto se ha adaptado para que simplemente genere como salida la etiqueta correspondiente a la entrada del usuario.

Estos modelos se compararán para determinar cuál proporciona mejores resultados en la clasificación de intenciones musicales, evaluando la capacidad de los modelos para interpretar y clasificar correctamente las intenciones del usuario.

## 2.6. Aprendizaje por Transferencia

El aprendizaje por transferencia es una estrategia dentro del campo del aprendizaje automático que consiste en tomar un modelo previamente entrenado y adaptarlo a una tarea nueva y más específica. El modelo se beneficia de lo aprendido anteriormente, trasladando esa experiencia a la resolución de problemas similares. Según Torrey y Shavlik, este enfoque intenta que los algoritmos de aprendizaje automático se aproximen a la eficiencia del aprendizaje humano, ya que las personas aplican de manera natural lo aprendido en situaciones anteriores cuando se enfrentan a retos nuevos [21].

Este enfoque ha revolucionado áreas como el procesamiento del lenguaje natural y la visión artificial. Modelos como BERT, RoBERTa o GPT aprenden patrones generales del lenguaje y se ajustan mediante técnicas como el *fine-tuning* para resolver tareas concretas. Gracias a esta técnica, es posible beneficiarse del potencial del aprendizaje profundo sin necesidad de contar con grandes recursos o volúmenes de datos en cada proyecto.



Figura 2.10: Proceso de aprendizaje por transferencia

### 2.6.1. Fine-tuning y optimización de hiperparámetros

El ajuste fino, o *fine-tuning*, es una técnica fundamental dentro del *deep learning*, especialmente en el contexto del aprendizaje por transferencia. Consiste en reutilizar un modelo preentrenado en una tarea general para adaptarlo a una tarea más específica, realizando un entrenamiento sobre un nuevo conjunto de datos más reducido y orientado al problema concreto.

Entrenar un modelo de cero requiere grandes cantidades de datos, recursos computacionales y largos tiempos de entrenamiento. Por ello, el *fine-tuning* es una estrategia muy eficiente, ya que partir de un modelo entrenado hace que el proceso de especialización de nuestro modelo se vuelva mucho más rápido y accesible [22].

Junto al *fine-tuning*, se realiza la optimización de hiperparámetros que se trata de buscar los valores más adecuados para parámetros como la tasa de aprendizaje, el tamaño de lote, el coeficiente de regularización o la proporción de calentamiento. Para ello se ha utilizado Optuna, que automatiza este proceso y nos permite obtener la combinación de hiperparámetros más óptima que mejora el rendimiento del modelo.

## 2.7. Ensemble de Modelos

En este proyecto se ha implementado una estrategia sencilla de ensemble basada en votación mayoritaria. Esta técnica del aprendizaje automático consiste en combinar las predicciones de los modelos implementados para obtener un resultado más robusto y preciso que el que se lograría con un solo modelo.

Durante la inferencia, cada modelo realiza su propia predicción de clase para una misma entrada y la predicción final será la clase más repetida entre las tres salidas. Este ensemble es catalogado como "no ponderado" ya que todos los modelos participantes tienen la misma importancia en el proceso de decisión (Figura 2.11).

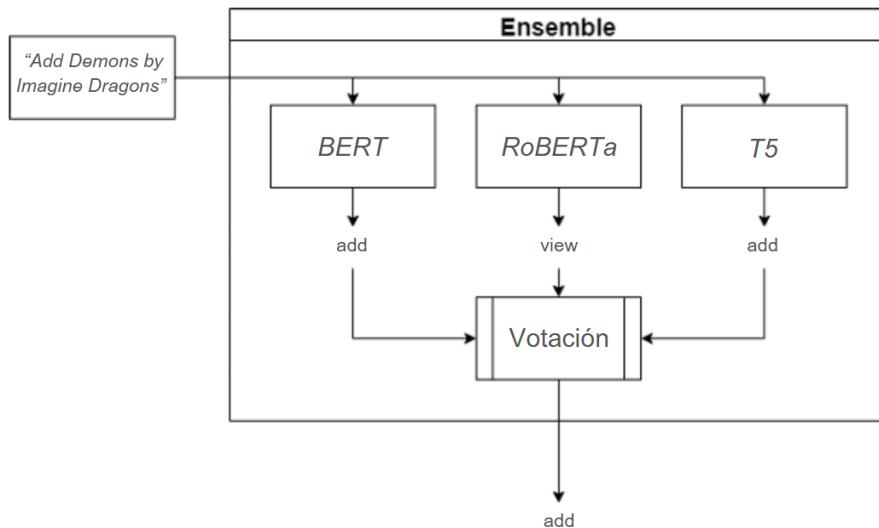


Figura 2.11: Funcionamiento de un ensemble de modelos

## 2.8. Medidas de Evaluación

Para evaluar los modelos de clasificación desarrollados en este trabajo se utilizan métricas estándar del aprendizaje automático, adaptadas a problemas de clasificación multiclas. Estas métricas nos permiten medir cuántas veces acierta el modelo y también detectar en qué aspectos concreta sus errores. En concreto, se han empleado cuatro indicadores principales: *accuracy*, *precision*, *recall* y *F1-score*.

Para poder definir correctamente estas métricas, conviene introducir primero un concepto fundamental: la **matriz de confusión** (Tabla 2.1). Esta herramienta nos permite analizar cómo se comporta un modelo de clasificación. Se trata de una tabla que muestra cuántas veces el modelo predijo correctamente cada clase (verdaderos positivos y negativos) y cuántas veces se confundió con otras (falsos positivos y falsos negativos).

	Predictión positiva	Predictión negativa
Clase real positiva	Verdadero Positivo (TP)	Falso Negativo (FN)
Clase real negativa	Falso Positivo (FP)	Verdadero Negativo (TN)

Tabla 2.1: Matriz de confusión binaria utilizada para definir las métricas de evaluación.

Aunque la tabla anterior (Tabla 2.2) se refiere a un caso de clasificación binaria, estos conceptos pueden extenderse a tareas de clasificación multiclas como la de este proyecto. En este caso, lo que se hace es aplicar la lógica de la matriz de confusión a cada clase por separado, es decir, se considera una clase como “positiva” y todas las demás como “negativas”, calculando así los valores de TP, FP, FN y TN para cada una (Tabla 2.2). De esta forma, se obtiene una métrica de *precision*, *recall* y *F1-score* por clase y se hace la media para obtener una visión global del rendimiento del modelo.

Clase real \ Predicción	add	remove	view	clear
add	TP <sub>add</sub>	FP	FP	FP
remove	FP	TP <sub>remove</sub>	FP	FP
view	FP	FP	TP <sub>view</sub>	FP
clear	FP	FP	FP	TP <sub>clear</sub>

Tabla 2.2: Matriz de confusión multiclas entre las intenciones predichas y las reales.

Como puede observarse en la Tabla 2.2, una matriz de confusión multiclas permite visualizar no solo los aciertos del modelo para cada intención, sino también los errores más frecuentes. A partir de esta base, podemos calcular las distintas métricas con las que mediremos el rendimiento del modelo.

### 2.8.1. Accuracy, Precision, Recall y F1-Score

Una vez introducida la matriz de confusión, podemos definir las métricas utilizadas para evaluar el rendimiento de nuestro modelo:

- **Accuracy:** indica el porcentaje de predicciones correctas sobre el total de ejemplos.

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FT} + \text{FN}} \quad (2.2)$$

- **Precision:** mide cuántas de las predicciones positivas fueron correctas.

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \quad (2.3)$$

- **Recall:** mide el porcentaje de ejemplos reales positivos que fueron correctamente identificados.

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \quad (2.4)$$

- **F1-score:** combina precisión y recall en un único valor, a través de su media armónica.

$$\text{F1-score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \quad (2.5)$$

### 2.8.2. Trabajos Relacionados

En esta sección se describen algunos proyectos representativos de asistentes conversacionales musicales basados en modelos de lenguaje y técnicas de NLP, cuyos enfoques inspiran y contrastan con la solución propuesta en este trabajo.

**MuseChat** [23]. MuseChat integra un gran modelo de lenguaje (*Vicuna-7B*, derivado de *LLaMA*) con un motor de recomendación musical orientado a contenido de vídeo. Su arquitectura consta de un módulo de recomendación musical, que sugiere canciones apropiadas para un vídeo dado, considerando el contexto y las preferencias del usuario. También tiene otro de razonamiento, que emplea un modelo de lenguaje grande para generar explicaciones en lenguaje natural sobre por qué se recomendó cierta canción.

**Chatbot Song Recommender System** [24]. Este proyecto presenta un flujo conversacional diseñado para capturar y refinar las preferencias del usuario. A partir de declaraciones como “Me gusta Coldplay”, el sistema utiliza técnicas de NLP para extraer entidades (artista, género, época) y aplica un algoritmo híbrido para generar playlists personalizadas. La interactividad se refuerza permitiendo al usuario ajustar en tiempo real criterios de recomendación, lo que demuestra la eficacia de combinar Transformers para comprensión de intenciones con métodos clásicos de recomendación.

**Music&Me Chatbot** [25]. Este proyecto es una aplicación web que facilita la interacción del usuario con un chatbot llamado Alex. Su principal característica es la recomendación de música basada en el análisis de las emociones del usuario durante la conversación. Mediante *IBM Tone Analyzer*, detecta el estado de ánimo del usuario a partir del texto de la conversación y utiliza la API de *Last.fm* para recomendar canciones acordes a ese tono emocional.

## 2.9. Tecnologías y Recursos Utilizados

Para el desarrollo de este trabajo se han utilizado diversas tecnologías, librerías y plataformas que han facilitado tanto la implementación de los modelos como la gestión del entorno

de trabajo. A continuación, se detallan las principales herramientas empleadas:

- **Python:** Lenguaje de programación con el que se ha desarrollado todo el proyecto, desde el preprocesamiento de datos y el entrenamiento de los modelos hasta el análisis de resultados y la implementación de la lógica del agente conversacional.
- **PyTorch:** Es una librería de aprendizaje automático utilizado para construir, entrenar y evaluar los modelos basados en Transformers.
- **Scikit-learn:** Empleada para tareas auxiliares como convertir etiquetas en números o dividir los datos en conjuntos de entrenamiento y validación.
- **Hugging Face:** Es una comunidad de código abierto que ha sido clave para cargar modelos preentrenados como *BERT*, *RoBERTa* y *T5*. Gracias a ello, se ha facilitado el acceso a modelos potentes sin necesidad de entrenarlos desde cero.
- **Google Colab:** Ha sido la plataforma de ejecución durante gran parte del proyecto. Al estar en la nube y ofrecer acceso gratuito a GPU, ha permitido entrenar modelos sin necesidad de tener un equipo propio potente.
- **Google Drive:** Para guardar los modelos entrenados, los registros y los resultados, facilitando el trabajo en diferentes sesiones de Colab.
- **Optuna:** Es una herramienta de optimización que ha servido para ajustar automáticamente los hiperparámetros del entrenamiento. En vez de probar valores manualmente, Optuna ha buscado las mejores combinaciones de forma eficiente.
- **Weights & Biases (W&B):** Es una plataforma con herramientas que nos ha permitido realizar el seguimiento de los experimentos, permitiendo registrar, visualizar y comparar el rendimiento de los modelos a lo largo del proceso de entrenamiento.
- **Spotify API:** Para obtener los datos de las canciones solicitadas por el usuario, se ha utilizado la API pública de Spotify. Esta interfaz nos ha permitido acceder a datos actualizados sobre títulos, artistas, álbumes o géneros musicales, lo que ha sido muy útil para implementar funciones como la búsqueda y recomendación de canciones.
- **Interfaz web (Node.js + JavaScript):** La interfaz gráfica del sistema ha sido desarrollada con tecnologías web estándar (HTML, CSS y JavaScript) y ejecutada mediante Node.js, que actúa como servidor de desarrollo para lanzar la aplicación en local.
- **Socket.IO:** Biblioteca que facilita la comunicación bidireccional en tiempo real entre cliente y servidor, utilizada para transmitir los mensajes del usuario y las respuestas del bot de forma instantánea.
- **React:** Biblioteca de JavaScript para construir interfaces de usuario reactivas y dinámicas, empleada en el frontend del chatbot para gestionar el estado y renderizar los componentes de la conversación.

## 2. Marco Teórico

---

- **Overleaf:** Plataforma online para la edición de documentos en LaTeX. Se ha utilizado para redactar y maquetar esta memoria.



Figura 2.12: Tecnologías y recursos utilizados en el desarrollo del trabajo

En resumen, este capítulo presenta los fundamentos teóricos necesarios para entender los enfoques adoptados en este trabajo. Estos conceptos me han servido para comprender las decisiones técnicas y metodológicas que se presentan en los capítulos siguientes.

## CAPÍTULO 3

# Metodología, Experimentación y Resultados

---

Este capítulo describe el proceso seguido para llevar a cabo el desarrollo del Chatbot, desde la preparación de los datos hasta la evaluación final de los modelos. Se explican los pasos concretos realizados en cada fase del desarrollo, incluyendo la selección de modelos, el ajuste de hiperparámetros y la estrategia de evaluación.

### 3.1. Descripción General de la Metodología

El desarrollo del sistema conversacional propuesto en este trabajo se ha llevado a cabo siguiendo una metodología estructurada en distintas fases:

- **Preprocesamiento de los datos:** en esta fase se etiquetaron las intenciones, y se prepararon las entradas para ser procesadas por los modelos. También se llevó a cabo la tokenización y la codificación necesaria para el entrenamiento.
- **Selección y definición del modelo:** se eligieron tres modelos de lenguaje basados en arquitecturas Transformer (*BERT*, *RoBERTa* y *T5*) para abordar la tarea de clasificación según las intenciones del usuario.
- **Ajuste de hiperparámetros:** se empleó la herramienta Optuna para explorar distintas combinaciones de hiperparámetros y seleccionar aquellas que ofrecieran un mejor equilibrio entre precisión y generalización.
- **Entrenamiento del modelo:** una vez definidos los modelos y sus parámetros óptimos, se procedió a su entrenamiento utilizando un conjunto de datos previamente dividido.
- **Evaluación y análisis de resultados:** finalmente, se evaluó el rendimiento de cada modelo empleando métricas estándar como accuracy, precision, recall y F1-score. Los resultados obtenidos permitieron comparar las diferentes arquitecturas y justificar la elección final.

Para facilitar la comprensión del proceso completo, en la Figura 3.1 se incluye un esquema general que resume las fases seguidas en el desarrollo experimental del proyecto.

### 3. Metodología, Experimentación y Resultados

---

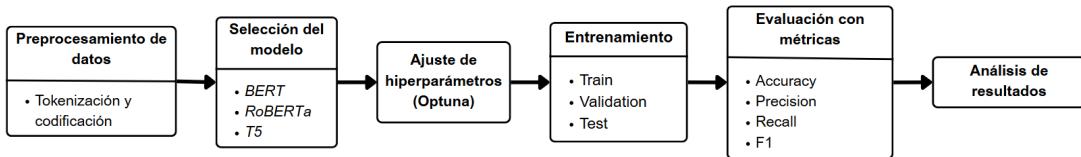


Figura 3.1: Diagrama general del flujo de trabajo seguido en el proyecto.

## 3.2. Preparación de los Datos

Antes de entrenar cualquier modelo, es necesario preparar correctamente los datos que se van a utilizar. En este trabajo, el conjunto de datos ha sido adaptado para que pueda ser interpretado por los modelos de lenguaje, lo que ha requerido una serie de transformaciones previas. En los apartados siguientes se detalla todo este proceso.

### 3.2.1. Descripción de los Datos

El conjunto de datos utilizado en este proyecto está compuesto por frases escritas en lenguaje natural, simulando expresiones que el usuario podría emplear al interactuar con un asistente musical. Estas frases fueron generadas mediante un modelo de inteligencia artificial y posteriormente revisadas manualmente para asegurar su coherencia. Cada frase está etiquetada con una de las cuatro intenciones principales del sistema: `add`, `remove`, `view` y `clear`, que hacen referencia a acciones sobre una lista de reproducción.

Muchas de las frases contienen errores ortográficos, expresiones informales o repeticiones innecesarias que imitan la forma en que un usuario real podría comunicarse. A continuación, se muestran algunos ejemplos representativos:

- “add Believer by Imagine Dragons to my playlist bro”      **Intención: add**
- “Remov Viva la vida, pls”      **Intención: remove**
- “heyy, what’s on my playlist right now?”      **Intención: view**
- “Erase evrything u have, start from scratch”      **Intención: clear**

Este tipo de frases hace que el problema sea más realista y obliga a los modelos a generalizar más allá de estructuras gramaticales correctas.

Los datos se encuentran organizados en tres archivos en formato CSV, cada uno con dos columnas: `text`, que contiene la instrucción del usuario, e `intent`, que indica la clase correspondiente. El conjunto de entrenamiento está formado por 1960 ejemplos (70 %), distribuidos de forma equilibrada con 490 frases por clase. Los archivos de validación y

test tienen 420 (15 %) entradas cada uno, con 105 frases para cada intención. Se garantizó el equilibrio de clases en cada subconjunto mediante muestreo estratificado.

En la Figura 3.2 se muestra la distribución de etiquetas en los tres subconjuntos del corpus. Esta distribución equilibrada resulta especialmente útil para evitar sesgos durante el entrenamiento y facilita una evaluación justa entre clases.

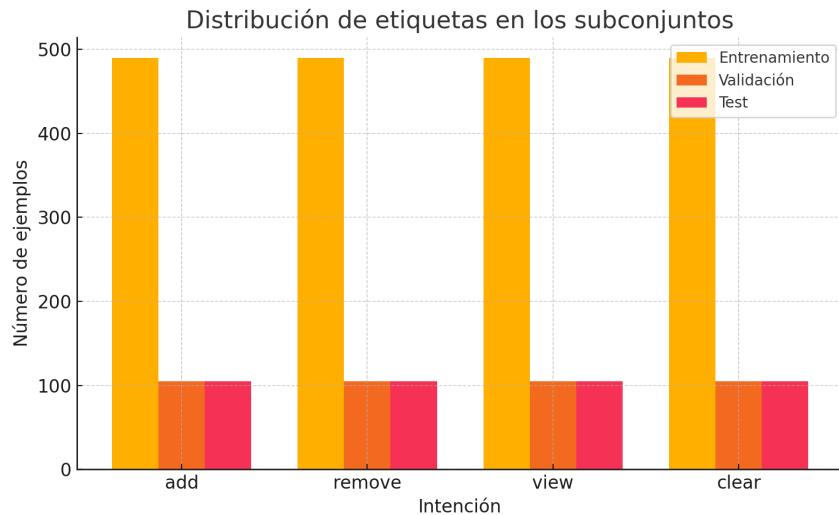


Figura 3.2: Distribución de ejemplos por intención en los subconjuntos de entrenamiento, validación y test.

#### 3.2.2. Limpieza y Normalización

En este proyecto no se ha llevado a cabo un proceso de limpieza o normalización del texto, como eliminar signos de puntuación, convertir a minúsculas o corregir errores ortográficos. Nos encontramos ante un entorno conversacional real donde un usuario puede expresarse de manera coloquial y con todo tipo de errores gramaticales. Por lo tanto, se busca que el modelo aprenda a lidiar con entradas realistas y ruidosas, tal como se espera en una interacción natural con usuarios. Además, los modelos utilizados están preentrenados sobre corpus amplios que ya incluyen variabilidad textual y son capaces de manejar este tipo de entradas sin necesidad de una normalización previa.

#### 3.2.3. Tokenización y Codificación

Una vez preparados los datos, el siguiente paso consiste en convertir las frases de texto en una representación numérica que pueda ser interpretada por el modelo. Para realizar este proceso, hacemos uso de un *tokenizador* que divide el texto en *tokens*.

En este proyecto se ha utilizado el tokenizador correspondiente a cada modelo selec-

### **3. Metodología, Experimentación y Resultados**

---

cionado, ya que cada arquitectura ha sido preentrenada con un esquema específico de tokenización. Estos tokenizadores no solo dividen el texto, sino que también lo codifican en una secuencia de identificadores numéricos (`input_ids`), que son los valores que realmente se introducen en el modelo.

#### *Ejemplo de tokenización con BERT/RoBERTa*

Dada la frase de entrada: “add Believer to my playlist, pls”

La tokenización produce los siguientes tokens:

```
[‘add’, ‘Believer’, ‘to’, ‘my’, ‘play’, ‘##list’, ‘pls’]
```

Y su correspondiente codificación numérica (`input_ids`):

```
[12345, 6789, 234, 3456, 2525, 9876, 5432]
```

(Los números anteriores son ficticios, generados solo a modo de ejemplo ilustrativo).

Para el modelo T5, se utiliza un esquema de tokenización distinto a *BERT* y *RoBERTa*. Estos últimos emplean el algoritmo **WordPiece**, mientras que T5 utiliza **SentencePiece**. WordPiece necesita preprocessar el texto separándolo por espacios, utilizando un prefijo especial (`##`) para indicar que un fragmento va en medio de una palabra. SentencePiece trata toda la entrada como una cadena continua y codifica también los espacios (indicados con `_`) [26].

#### *Ejemplo de tokenización con T5*

Dada la frase de entrada: “I want to see my playlist”

La tokenización produce los siguientes tokens:

```
[‘_I’, ‘_want’, ‘_to’, ‘_see’, ‘_my’, ‘_playlist’]
```

Y su correspondiente codificación numérica (`input_ids`):

```
[121, 297, 12, 1144, 102, 15007]
```

(Los números anteriores son ficticios, generados solo a modo de ejemplo ilustrativo).

Además, al tratarse de un modelo generativo tipo encoder-decoder, su salida no es una clase numérica directa, sino una secuencia textual que representa la etiqueta deseada (por ejemplo, “add” o “view”). Por este motivo, en este caso se prepararon los datos en formato instrucción-respuesta, donde el texto del usuario se presenta como un comando y el modelo genera la intención como texto.

### **3.3. Selección de los Modelos**

A continuación se describen los modelos seleccionados para llevar a cabo la experimentación en este trabajo. Todos los modelos son pre-entrenados y se les ha aplicado la técnica de

ajuste fino (fine-tuning) utilizando nuestro conjunto de datos.

***bert-base-uncased***<sup>1</sup>: BERT fue pre-entrenado con grandes cantidades de texto y tiene la capacidad de comprender el contexto de las palabras. Se utilizó la versión uncased, lo que significa que BERT no diferencia entre mayúsculas y minúsculas, permitiendo una mayor generalización en los datos.

***roberta-base***<sup>2</sup>: es una variante optimizada de BERT que ha demostrado un rendimiento superior en múltiples tareas de NLP. La versión base de RoBERTa se eligió por su eficiencia y por ser efectiva en tareas de clasificación de texto.

***t5-small***<sup>3</sup>: T5 transforma todas las tareas de lenguaje natural en un problema de entrada-salida de texto a texto. En este proyecto, se utilizó la versión pequeña de T5, ya que proporciona un buen equilibrio entre eficiencia y rendimiento.

## 3.4. Ajuste de Hiperparámetros

El ajuste de hiperparámetros es una etapa clave en el desarrollo de modelos de aprendizaje automático, ya que estos valores influyen directamente en la calidad del entrenamiento y en el rendimiento final.

### 3.4.1. Métodos de Búsqueda

Para optimizar el rendimiento de los modelos, se llevó a cabo una fase de ajuste de hiperparámetros utilizando el framework **Optuna**. Gracias a esta herramienta podemos automatizar la búsqueda de combinaciones óptimas de parámetros, evitando así la necesidad de explorar manualmente todas las posibilidades.

Optuna emplea una técnica llamada **optimización bayesiana basada en árboles de parzen** (TPE). Esta estrategia consiste en estimar un área prometedora guiándose por los resultados obtenidos en ensayos anteriores. Cada vez que prueba una combinación de hiperparámetros, guarda el resultado y lo utiliza para decidir qué valores probar a continuación, lo que le permite refinar progresivamente la región de búsqueda. Este enfoque iterativo hace que el proceso sea mucho más eficiente que probar combinaciones al azar o de forma exhaustiva [27].

---

<sup>1</sup><https://huggingface.co/google-bert/bert-base-uncased>

<sup>2</sup><https://huggingface.co/FacebookAI/roberta-base>

<sup>3</sup><https://huggingface.co/google-t5/t5-small>

### 3. Metodología, Experimentación y Resultados

---

#### 3.4.2. Hiperparámetros Evaluados

Durante el proceso de optimización se evaluaron distintos hiperparámetros clave que influyen directamente en el rendimiento del modelo. La selección de estos parámetros se basó en su impacto conocido sobre el entrenamiento y la generalización, especialmente en modelos de tipo Transformer. En la Tabla 3.2 se detallan los hiperparámetros ajustados, así como el espacio de búsqueda definido para cada uno de ellos en Optuna.

Hiperparámetro	Espacio de búsqueda
<code>learning_rate</code>	Log-uniforme en $[1 \cdot 10^{-6}, 5 \cdot 10^{-5}]$
<code>batch_size</code>	Valores categóricos: {8, 16, 32}
<code>weight_decay</code>	Uniforme en [0,01, 0,2]
<code>warmup_ratio</code>	Uniforme en [0,1, 0,3]

Tabla 3.1: Espacio de búsqueda definido para los hiperparámetros evaluados con Optuna.

#### 3.4.3. Tamaño del Conjunto de Datos Utilizado

Para la búsqueda de hiperparámetros se ha utilizado el conjunto completo de entrenamiento, compuesto por 1960 entradas distribuidas de forma equilibrada entre las cuatro clases. Al tratarse de un dataset pequeño y balanceado, no fue necesario realizar particiones adicionales ni reducir el volumen de datos. Esta decisión permite que la evaluación de cada combinación de hiperparámetros se realice con la mayor cantidad de información posible.

#### 3.4.4. Criterios de Selección

Para la selección de la combinación óptima de hiperparámetros, optamos por usar la métrica `eval_loss` (pérdida de evaluación) durante la optimización.

En etapas iniciales de experimentación, consideramos `eval_accuracy` como métrica principal para la optimización, ya que buscábamos principalmente que el modelo asignara correctamente la etiqueta correspondiente a la entrada del usuario la mayor cantidad de veces posible. Sin embargo, al optimizar `eval_loss` descubrimos que los modelos conseguían un `accuracy` prácticamente idéntico al obtenido al optimizar directamente sobre `eval_accuracy`.

Por tanto, entrenamos nuestros modelos evaluando siempre su `eval_loss` y seleccionamos para el ensemble aquellos checkpoints de cada modelo que presentaron la menor pérdida en validación. De este modo, nos aseguramos de incorporar al sistema final los tres modelos más robustos frente al sobreajuste .

## 3.5. Entrenamiento de los Modelos

En esta sección se describirá cómo se llevó a cabo el entrenamiento de los tres modelos seleccionados. Se detallan las librerías empleadas, los valores finales de los hiperparámetros, el número de épocas y el uso de técnicas como *early stopping*. El objetivo es mostrar paso a paso la configuración que nos permitió obtener el mejor rendimiento en validación y comparar los modelos.

### 3.5.1. Configuración de Entrenamiento

El entrenamiento se llevó a cabo en un nodo equipado con GPU Tesla T4 sobre Google Colab Pro, utilizando Python, PyTorch y la librería `Transformers`. Para la optimización de hiperparámetros se utilizó Optuna, y para el cálculo de métricas se emplearon las funciones de `scikit-learn`.

Tras la tokenización y el preprocesamiento de los datos, hemos empleado la clase `Trainer` de la librería `Transformer` para el entrenamiento de los modelos. La configuración de entrenamiento se centró en los siguientes hiperparámetros clave:

- **Número de épocas** (`num_train_epochs`): 10
- **Métrica para seleccionar el mejor modelo** (`metric_for_best_model`): `eval_loss` (pérdida de evaluación).
- **Dropout**: 0.3, para desactivar aleatoriamente un porcentaje de las neuronas.
- **Estrategia de evaluación** (`eval_strategy`): "epoch", para que el modelo se evalue al final de cada época.

Además, los mejores valores de hiperparámetros obtenidos mediante la optimización con Optuna para cada uno de los modelos evaluados se detallan en la Tabla 3.2

Modelo	Learning Rate	Batch Size	Weight Decay	Warmup Ratio
BERT	$1,96 \times 10^{-5}$	8	0.1722	0.1429
RoBERTa	$3,69 \times 10^{-5}$	32	0.1579	0.1230
T5	$2,14 \times 10^{-5}$	8	0.0851	0.1149

Tabla 3.2: Hiperparámetros óptimos encontrados por Optuna para cada modelo.

### 3.5.2. Monitorización del Entrenamiento

El entrenamiento de los modelos fue monitorizado usando Weights & Biases (W&B) para visualizar en tiempo real las métricas de entrenamiento. Esta plataforma nos ofreció un

### 3. Metodología, Experimentación y Resultados

---

panel interactivo que facilitó el seguimiento de la *Training Loss*, la *Validation Loss* y otras métricas de evaluación a lo largo de las épocas.

En nuestro caso nos hemos centrado en analizar la pérdida (*loss*), ya que nos interesa conocer la rapidez de convergencia de cada modelo, su estabilidad en validación y la detección temprana de sobreajuste. En conjunto, las tres arquitecturas muestran una rápida convergencia y un comportamiento estable en validación:

- **BERT-base:** la pérdida de ambos conjuntos desciende rápidamente y convergen alrededor de la época 4 y las curvas de entrenamiento y validación evolucionan de forma casi paralela en las últimas épocas (Figura 3.3).
- **RoBERTa-base:** se observa que reduce rápidamente su pérdida de validación de 5.2 a 0.7 en cinco épocas y, a partir de la época 6, ambas curvas descienden en paralelo con la validación siempre ligeramente por debajo. Este comportamiento nos indica que no hay sobreajuste y que la regularización interna favorece la generalización. (Figura 3.4).
- **T5-small:** converge todavía más rápido, alcanzando pérdidas de validación cercanas a cero ya en la época 3 y manteniéndose prácticamente nula hasta la época 10. La evolución paralela de ambas curvas nos indica la ausencia de sobreajuste (Figura 3.5).

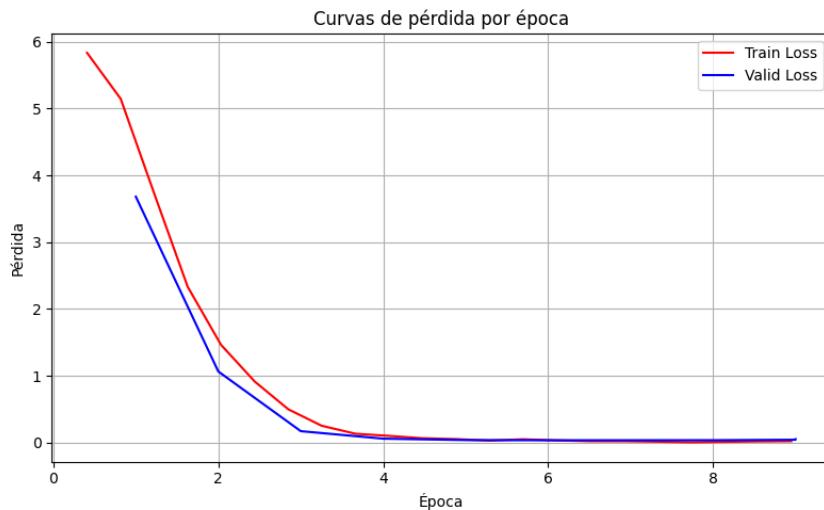


Figura 3.3: Curvas de pérdida por época para BERT (bert-base-uncased).



Figura 3.4: Curvas de pérdida por época para RoBERTa (roberta-base).

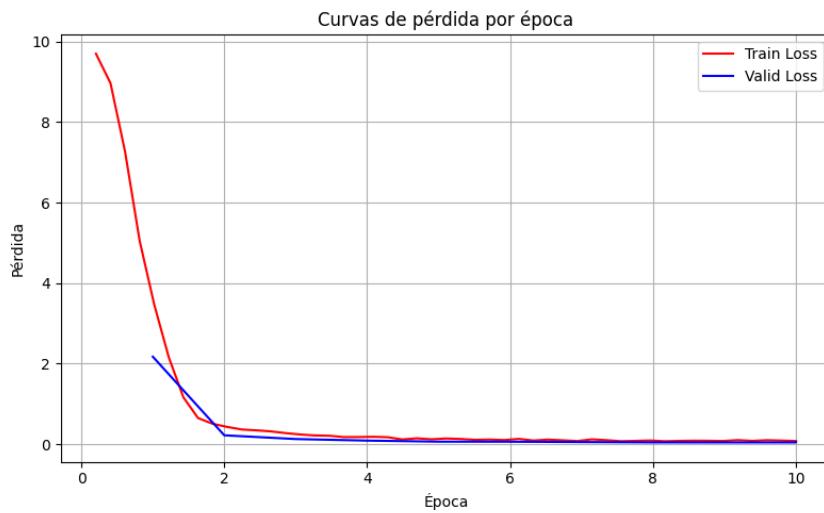


Figura 3.5: Curvas de pérdida por época para T5 (t5-small).

En todos los casos se configuró un `EarlyStoppingCallback` con paciencia de tres épocas. En los entrenamientos de RoBERTa y T5 el proceso llegó hasta la décima época, puesto que la pérdida seguía descendiendo mínimamente, pero en el caso de BERT-base el entrenamiento se detuvo automáticamente en la época 9 al no observarse mejora en validación durante tres iteraciones consecutivas. De este modo, el checkpoint final de cada modelo corresponde al que mejor se generaliza al conjunto de validación, garantizando un buen rendimiento a la hora de evaluar en datos nuevos.

### 3.6. Evaluación y Análisis de Resultados

En este apartado se presentan los resultados obtenidos sobre el conjunto de test para las tres arquitecturas, se analizan sus métricas cuantitativas y se profundiza en la matriz de confusión para comprender mejor los patrones de error.

En la Tabla 3.3 se presentan los resultados globales obtenidos por los modelos entrenados sobre el conjunto de test. Se incluyen las métricas de accuracy, precisión, recall y F1-score.

Modelo	Accuracy	Precision	Recall	F1-score
BERT-base	<b>0.9905</b>	<b>0.9905</b>	<b>0.9905</b>	<b>0.9905</b>
RoBERTa-base	0.9762	0.9762	0.9762	0.9762
T5-small	0.9595	0.9622	0.9595	0.9600

Tabla 3.3: Resultados de evaluación sobre el conjunto de test

En la Tabla 3.4 se presentan los resultados obtenidos por los modelos entrenados sobre el conjunto de test desglosados por etiqueta.

Modelo	Clase	Precision	Recall	F1-score
BERT-base	add	0.9811	0.9905	0.9858
	remove	0.9904	0.9810	0.9857
	view	0.9906	1.0000	0.9953
	clear	1.0000	0.9905	0.9953
RoBERTa-base	add	0.9811	0.9905	0.9860
	remove	0.9717	0.9810	0.9765
	view	0.9714	0.9714	0.9714
	clear	0.9806	0.9619	0.9724
T5-small	add	1.0000	0.9810	0.9904
	remove	0.9608	0.9333	0.9469
	view	1.0000	0.9429	0.9706
	clear	0.8879	0.9810	0.9321

Tabla 3.4: Resultados por etiqueta para los modelos evaluados sobre el conjunto de test.

A partir de los resultados globales (Tabla 3.3) observamos que BERT-base obtiene la mayor precisión y balance entre métricas (accuracy, precision, recall y F1-score todas en 0.9905), seguido de RoBERTa-base ( $\approx 0.976$ ) y T5-small ( $\approx 0.960$ ). Sin embargo, el análisis por clase (Tabla 3.4) revela matrices interesantes:

- Para la intención `add`, los tres modelos obtienen F1 muy altos: BERT-base  $F1 = 0.9858$ , RoBERTa-base  $F1 = 0.9860$  y T5-small  $F1 = 0.9904$ , lo que evidencia que esta clase se detecta con excelente fiabilidad.
- En `remove`, BERT-base mantiene una F1 de  $F1 = 0.9857$  y RoBERTa-base  $F1 =$

0,9765, mientras que T5-small baja a  $F1 = 0,9469$ , indicando que el modelo generativo confunde algo más esta clase con otras.

- La categoría **view** presenta  $F1 = 0,9953$  en BERT-base,  $F1 = 0,9714$  en RoBERTa-base y  $F1 = 0,9706$  en T5-small, lo que muestra un rendimiento ejemplar de BERT y una ligera caída en los demás, aunque por encima de 0.97.
- Para **clear**, BERT-base alcanza  $F1 = 0,9953$ , RoBERTa-base  $F1 = 0,9724$  y T5-small  $F1 = 0,9321$ , siendo esta última la clase con mayor margen de mejora, probablemente por solapamientos léxicos con **remove**.

En conjunto, estos resultados confirman que los modelos basados en encoder (BERT y RoBERTa) superan ligeramente a T5-small en esta tarea de clasificación pura.

Los resultados obtenidos reflejan un rendimiento notable de los modelos. Al centrarnos en acciones concretas como 'añadir', 'eliminar', 'ver' y 'limpiar', con formulaciones breves y directas, se ha facilitado la identificación de patrones consistentes en el lenguaje de los usuarios.

Este enfoque controlado ha sido ideal para validar la capacidad de los modelos en escenarios con baja ambigüedad, sentando una base sólida para futuras aplicaciones más complejas. Si bien es esperable que el rendimiento pueda variar en contextos con mayor diversidad lingüística, los resultados actuales demuestran la efectividad del modelo en tareas de intención bien acotadas, lo cual cumple plenamente con los objetivos de este proyecto.

#### Matrices de Confusión

Para profundizar en los patrones de error de cada modelo, a continuación se presentan las matrices de confusión correspondientes al conjunto de test. Cada figura muestra la distribución de predicciones frente a etiquetas verdaderas para el modelo indicado.

En la Figura 3.6 se presenta la matriz de confusión para BERT-base, donde se observa una concentración casi total de aciertos en la diagonal principal. Esto manifiesta la excelente capacidad de BERT-base para discriminar las cuatro intenciones.

En la Figura 3.7 se muestra la matriz de confusión para RoBERTa-base, donde predominan los aciertos en la diagonal principal y solo se registran algunas confusiones puntuales. Esto confirma su elevada capacidad de generalización.

En la Figura 3.8 se muestra la matriz de confusión para T5-small, en la que, aunque acierta la mayoría de ejemplos, comete 7 errores al etiquetar **clear** como **remove** y 5 errores al clasificar **clear** también como **view**. Esto indica una tendencia del modelo a confundir diferentes acciones de borrado.

### 3. Metodología, Experimentación y Resultados

---

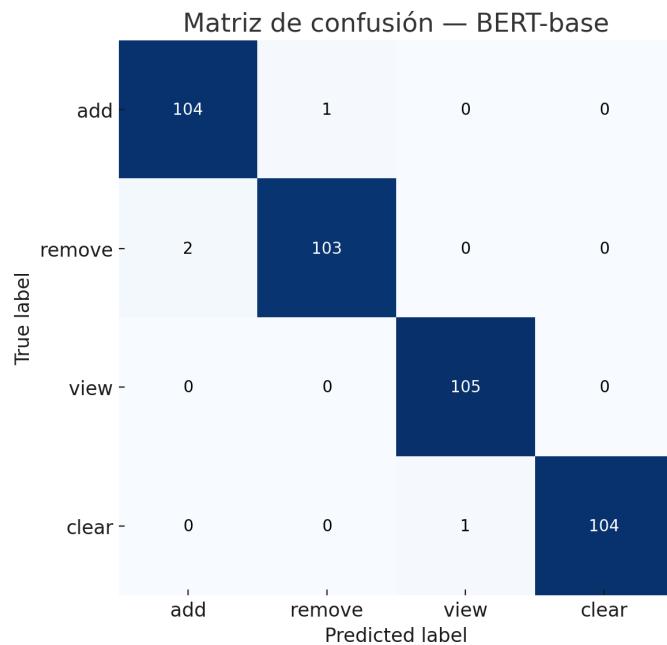


Figura 3.6: Matriz de confusión en test para BERT-base.

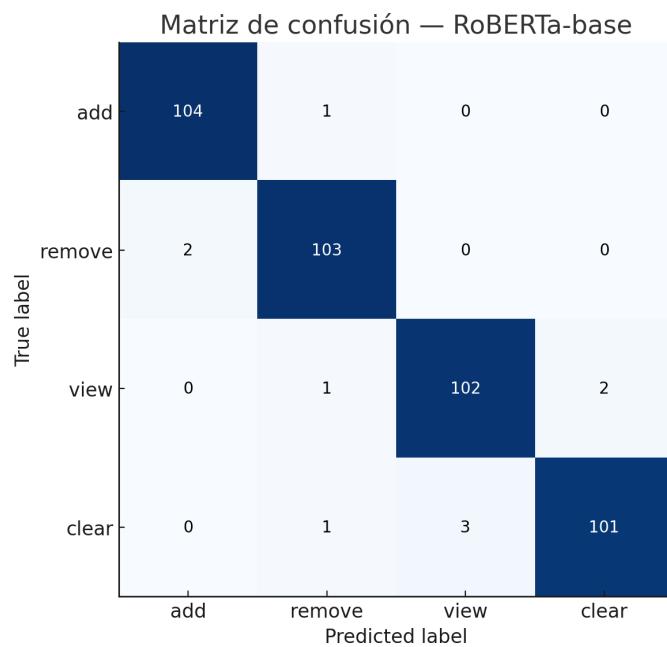


Figura 3.7: Matriz de confusión en test para RoBERTa-base.

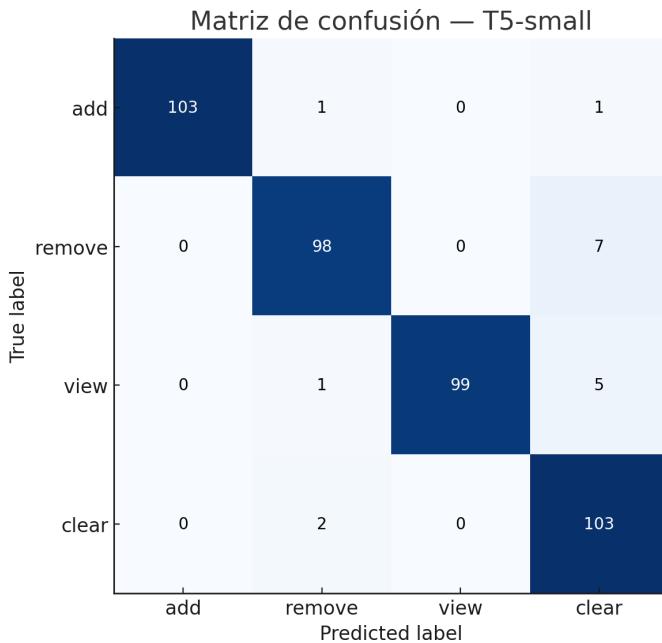


Figura 3.8: Matriz de confusión en test para T5-small.

En conjunto, las tres matrices de confusión confirman un rendimiento muy sólido. No obstante, BERT-base y RoBERTa-base superan ligeramente a T5-small en la distinción de comandos de borrado.

## 3.7. Análisis de Errores

En esta sección profundizamos en los fallos más representativos de los modelos, con el fin de identificar patrones sistemáticos y posibles causas de confusión. No se trata únicamente de cuantificar las métricas, sino de interpretar qué características de las entradas llevan al modelo a equivocarse.

### 3.7.1. Errores más frecuentes

El error más frecuente se produce al distinguir entre las intenciones `remove` y `clear`. Esta tendencia sugiere que en ciertos enunciados, especialmente aquellos que emplean sinónimos de “borrar”, la frontera entre estas dos clases no queda claramente definida.

#### Identificación de Errores Comunes

A continuación se presentan ejemplos representativos de entradas mal clasificadas.

### **3. Metodología, Experimentación y Resultados**

---

*Ejemplo 1:*

Texto: “please erase the track Yellow from my list”

Etiqueta real: remove Predicción: clear

El modelo interpretó “erase” como un comando de vaciado total (`clear`) al asociarlo con “my list” en lugar de reconocer el objeto específico. Para corregirlo, podría añadirse un paso de detección de objetos directos (“the track *X*”) que diferencie entre eliminar un elemento concreto y vaciar toda la lista.

*Ejemplo 2:*

Texto: “add Show Me Love by Robin to my playlist”

Etiqueta real: add Predicción: view

En este caso, el modelo interpretó “Show Me Love” como un comando de visualización (`view`) al confundir el verbo “show” con “view”.

#### **Reflexiones y Posibles Mejora**

A partir del análisis de errores y las confusiones observadas en las matrices de confusión, se extraen las siguientes reflexiones y propuestas de mejora para incrementar la robustez y precisión del sistema:

- **Aumentar el conjunto de datos:** Incluir expresiones más diversas para “remove” y “clear”.
- **Preprocesamiento basado en entidades:** Implementar un módulo de reconocimiento de entidades (artistas, álbumes, playlists) que etiquete automáticamente fragmentos del texto antes de la clasificación. De este modo, el modelo podrá diferenciar con mayor claridad cuando la acción afecta a un artista o a un elemento concreto frente a toda una colección.
- **Reglas de desambiguación sintáctica:** Añadir reglas lingüísticas que, por ejemplo, consideren que construcciones del tipo “remove/delete \* by \*” se corresponden con `remove`, mientras que comandos sin preposición de artista o que incluyan cuantificadores absolutos (“all”, “my entire”) se mapeen a `clear`.
- **Feedback y aprendizaje en línea:** Incorporar algún tipo de sistema en el chatbot que permita al usuario corregir la intención cuando el modelo falle, de modo que estas correcciones se utilicen como nuevos ejemplos de entrenamiento.

## 3.8. Implementación de Chatbot

En este apartado se describe la puesta en marcha del asistente musical, detallando la base de datos de canciones, la arquitectura general del sistema, la integración del modelo de clasificación y el funcionamiento del prototipo final.

La base del programa *Chat Widget*<sup>4</sup> fue proporcionada en la asignatura *Information Retrieval and Text Mining*<sup>5</sup> (DAT640) en la Universidad de Stavanger . Estos recursos proporcionados fueron fundamentales para el desarrollo inicial del chatbot, permitiendo la implementación de las funcionalidades clave y sirviendo de base para la personalización y expansión del sistema.

### 3.8.1. Base de Datos

Para obtener la información musical (títulos, artistas, álbumes), el sistema se conecta a la **API de Spotify**<sup>6</sup>. Esta API nos permite consultar metadatos de canciones y álbumes, así como generar recomendaciones basadas en ciertas características sonoras. La API de Spotify actúa como nuestra “base de datos” remota, evitando la necesidad de mantener un almacén local y garantizando datos siempre actualizados.

Por otro lado, se utiliza `sqlite3` para guardar las canciones que el usuario añade a su lista de reproducción. Este es un sistema de gestión de bases de datos ligero que permite almacenar datos localmente en un archivo único, de modo que, cada vez que se inicia el programa, la playlist contiene las canciones que se añadieron en la última sesión. Gracias a esta herramienta, conseguimos que el asistente tenga memoria para que las canciones guardadas permanezcan accesibles para el usuario incluso después de cerrar y volver a abrir el programa.

Columna	Tipo	Restricciones
id	INTEGER	PRIMARY KEY AUTOINCREMENT
spotify_id	TEXT	NOT NULL, UNIQUE
title	TEXT	NOT NULL
artist	TEXT	NOT NULL
album	TEXT	NOT NULL
genre	TEXT	(opcional)
release_date	TEXT	(opcional)

Tabla 3.5: Estructura de la tabla `user_songs`

<sup>4</sup><https://github.com/iai-group/ChatWidget?tab=readme-ov-file>

<sup>5</sup>[https://www.uis.no/en/course/DAT640\\_1](https://www.uis.no/en/course/DAT640_1)

<sup>6</sup><https://developer.spotify.com/documentation/web-api>

### 3. Metodología, Experimentación y Resultados

---

En la Tabla 3.5 se muestra la estructura de la tabla `user_songs`, que almacena de forma persistente las canciones que el usuario añade a su lista. Cada registro incluye un identificador interno (`id`), el `spotify_id` único de la pista en Spotify, y los metadatos básicos de la canción (`title`, `artist`, `album`). Además, se guardan opcionalmente el `genre` y la `release_date` para facilitar futuras recomendaciones.

#### 3.8.2. Arquitectura del Sistema

El chatbot está basada en una estructura cliente-servidor que se divide en dos componentes principales: el *frontend* y el *backend*.

El frontend (cliente) del chatbot está desarrollado utilizando HTML, CSS y React, una biblioteca de JavaScript ampliamente utilizada para crear interfaces de usuario interactivas. En este caso, React gestiona la interfaz donde los usuarios interactúan con el chatbot, escribiendo mensajes y recibiendo respuestas en tiempo real. El frontend también se ocupa de enviar las solicitudes de interacción al backend y presentar las respuestas de manera fluida.

El backend (servidor) del chatbot está implementado utilizando Node.js y Python. Node.js maneja la parte de la comunicación en tiempo real entre el cliente y el servidor de manera instantánea. Python se encarga de procesar los mensajes del usuario y gestionar la lógica necesaria para interactuar con la música, según las intenciones del usuario.

Para permitir la comunicación en tiempo real entre estos dos componentes, se ha utilizado Socket.io, una biblioteca de JavaScript que facilita la transmisión bidireccional de datos entre el cliente y el servidor de manera instantánea. Esta tecnología asegura que la conversación sea fluida y en tiempo real, permitiendo que los mensajes del chatbot sean procesados y respondidos inmediatamente.

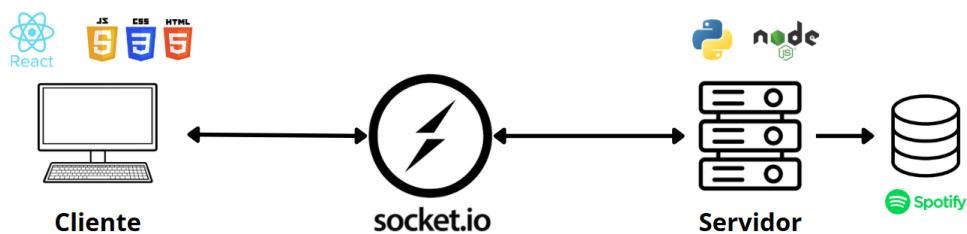


Figura 3.9: Arquitectura del Chatbot

En la Figura 3.9, se muestra la arquitectura del sistema, donde el frontend y el backend se comunican en tiempo real mediante Socket.io, utilizando React en el cliente y Node.js y Python en el servidor.

#### 3.8.3. Integración del Modelo

La integración del modelo en el chatbot implica cargar los modelos preentrenados y sus correspondientes tokenizadores. Estos modelos son responsables de procesar los mensajes recibidos, clasificando las intenciones del usuario y determinando la acción correspondiente.

Cuando el usuario interactúa con el chatbot, el mensaje se envía al backend, donde es tokenizado y procesado por los modelos para generar una predicción. Estas predicciones son utilizadas por el sistema para generar una respuesta que se envía al frontend en tiempo real. Esta integración permite que el chatbot responda de manera instantánea y precisa.

#### 3.8.4. Funcionalidades del sistema

El asistente ofrece cuatro operaciones principales basadas en las intenciones del usuario, más una funcionalidad de recomendación automática tras cada incorporación:

- **add:** el usuario puede añadir una canción de dos formas:
  - Indicando título y artista: por ejemplo “*Add Shape of You by Ed Sheeran*”. En este caso el bot añade directamente esa pista a la playlist.
  - Indicando solo el título: por ejemplo “*Add Hello*”. El bot muestra un listado de canciones que contienen ese título y solicita al usuario que elija el número correspondiente para añadir la pista deseada.

Tras confirmar la adición, el sistema genera automáticamente cinco recomendaciones basadas en la canción añadida: tres canciones del mismo artista y dos de género similar.

- **remove:** el usuario elimina una canción especificando siempre título y artista, por ejemplo “*Remove Hello by Adele*”. El bot confirma que la pista ha sido eliminada de la playlist.
- **view:** con este comando —por ejemplo “*Show my playlist*”— el bot muestra en pantalla la lista completa de canciones actualmente almacenadas en la playlist del usuario.
- **clear:** al recibir una petición como “*Clear my playlist*”, el bot vacía por completo la lista de reproducción y confirma que la playlist ha quedado limpia.

Esta estructura de intenciones permite una interacción intuitiva y flexible, cubriendo los casos de uso básicos de gestión de música y proporcionando recomendaciones contextuales que enriquecen la experiencia del usuario.

### **3. Metodología, Experimentación y Resultados**

---

#### **3.8.5. Resultado Final**

A continuación se muestran capturas de pantalla y ejemplos de uso del sistema:

En la Figura 3.10, se muestra el estado inicial del chatbot, donde el usuario puede ver las instrucciones para gestionar la lista de reproducción musical.

#### **Music Assistant System**

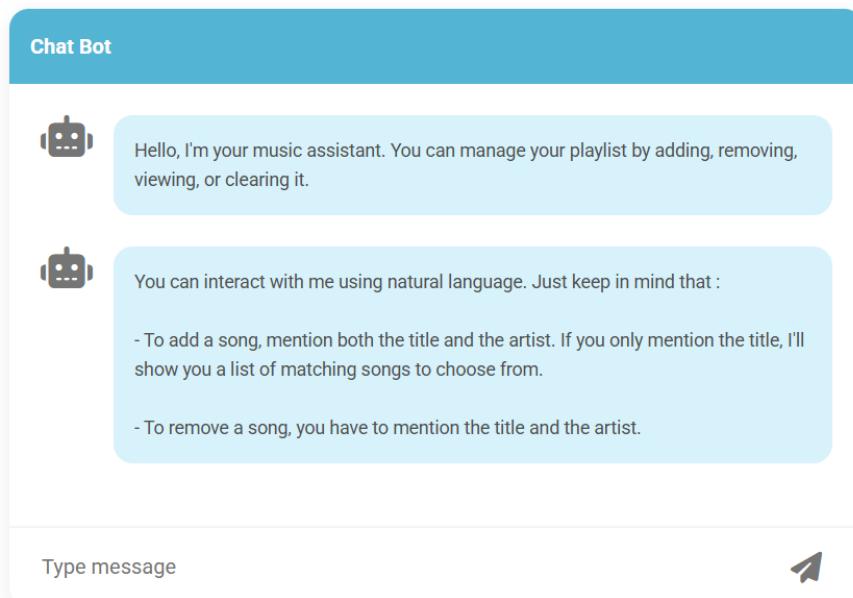


Figura 3.10: Captura de pantalla del chatbot al iniciar el programa.

En la Figura 3.11, se muestra cómo el usuario solicita añadir una canción indicando el título y el artista. Posteriormente, el bot recomienda canciones basadas en la última canción añadida, proporcionando sugerencias relacionadas con el artista y otras canciones similares del mismo género.

En la Figura 3.12 se muestra cómo el usuario solicita añadir una canción, pero en este caso, menciona únicamente el título de la canción. Esto provoca que el bot muestre una lista con algunas canciones que coinciden con ese título. El bot presenta las canciones encontradas junto con sus respectivos artistas y álbumes. Luego, el usuario puede seleccionar el número de la canción que desea añadir a su lista de reproducción, o ingresar 0 si no desea añadir ninguna canción.

En la Figura 3.13 se muestra cómo el usuario solicita ver su lista de canciones. En este caso, el bot responde mostrando las canciones que el usuario ha añadido previamente a su lista de reproducción, con detalles como el título, artista y álbum de cada canción.

## Music Assistant System

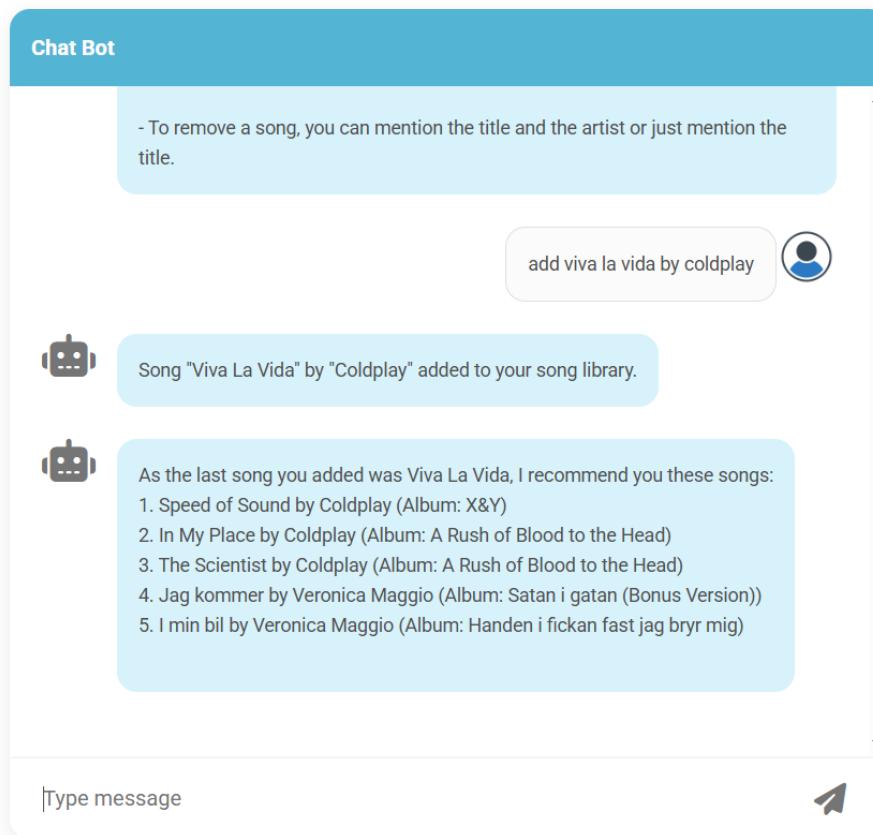


Figura 3.11: Captura de pantalla del chatbot al añadir una canción indicando el título y artista.

## Music Assistant System

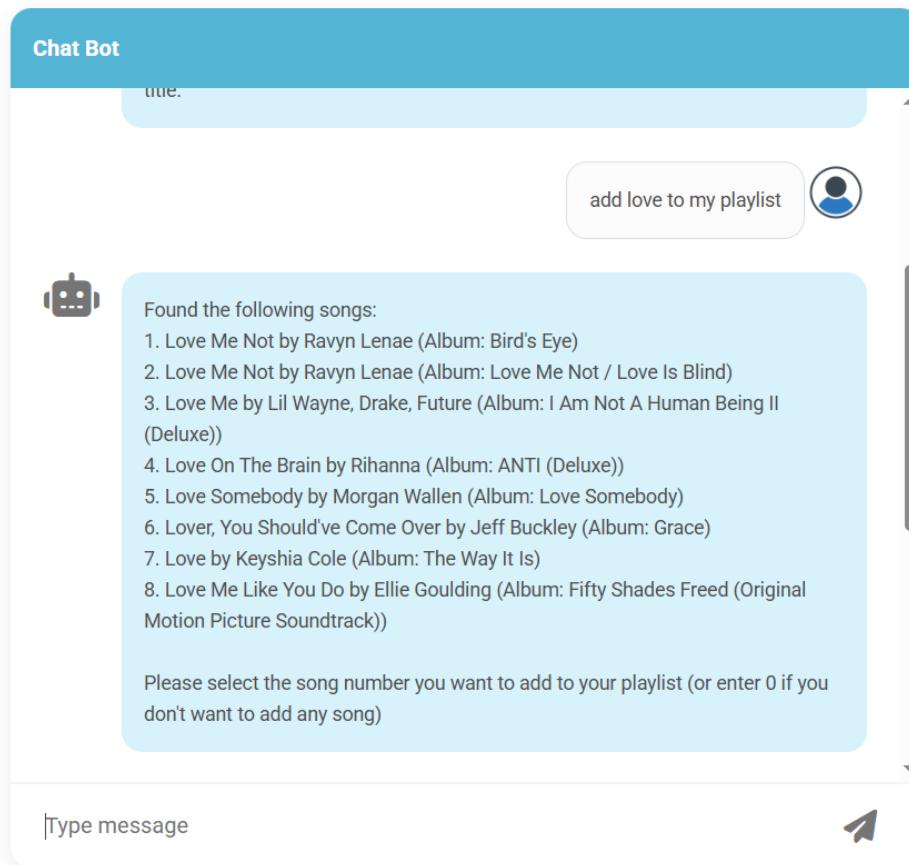


Figura 3.12: Captura de pantalla del chatbot al añadir una canción indicando únicamente el título.

## Music Assistant System

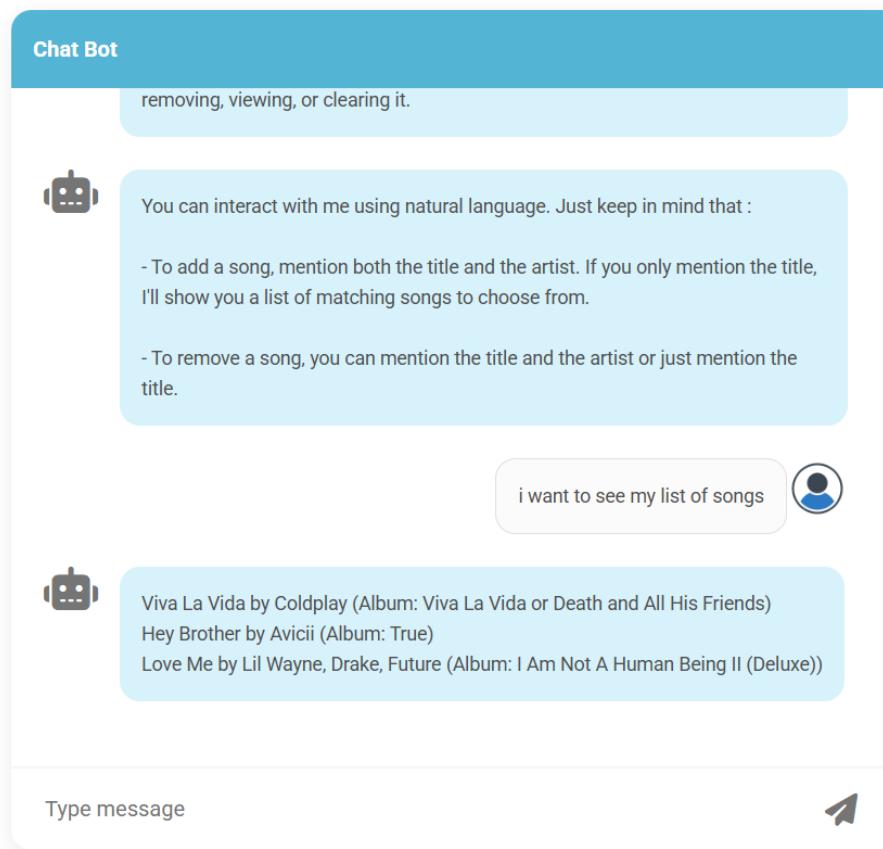


Figura 3.13: Captura de pantalla del chatbot al solicitar ver la playlist del usuario

### 3. Metodología, Experimentación y Resultados

En la Figura 3.14 se muestra cómo el usuario solicita la eliminación de una canción de la lista. El bot confirma que la canción ha sido eliminada correctamente con el mensaje. Para confirmarlo, se solicita ver la playlist, donde podemos apreciar que la canción mencionada no aparece.

## Music Assistant System

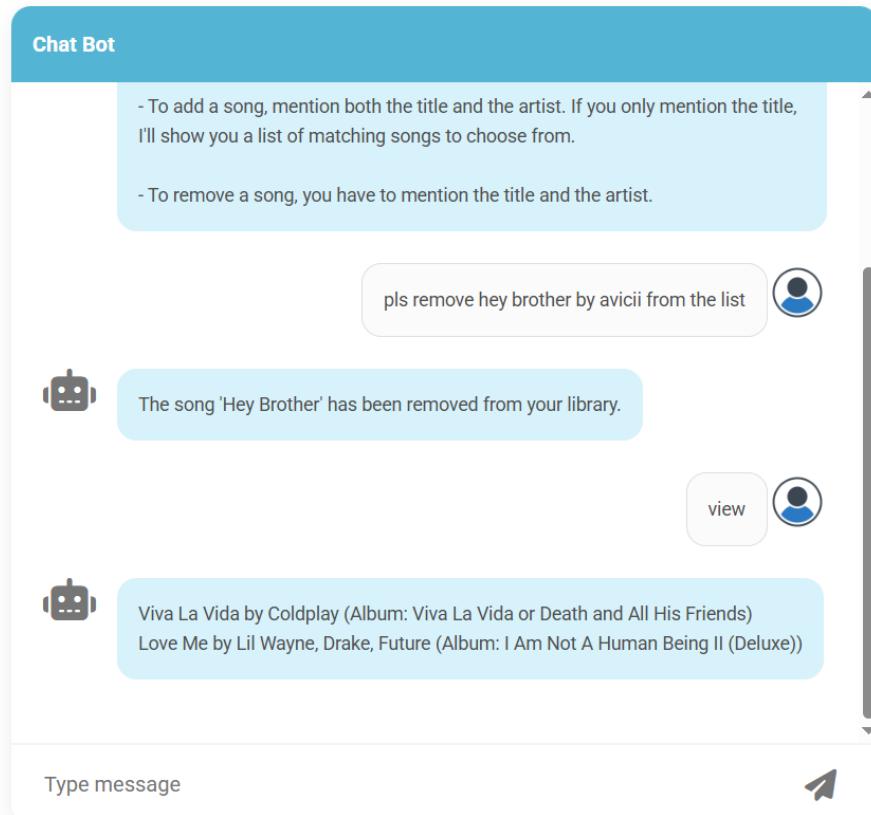


Figura 3.14: Captura de pantalla del chatbot al solicitar eliminar una canción de la playlist

En la Figura 3.15 se muestra cómo el usuario solicita vaciar por completo su lista de reproducción. El bot responde confirmando la acción. A continuación, se solicita ver la playlist para demostrar que todas las canciones han sido eliminadas correctamente y que el estado del playlist se ha actualizado en tiempo real.

## Music Assistant System

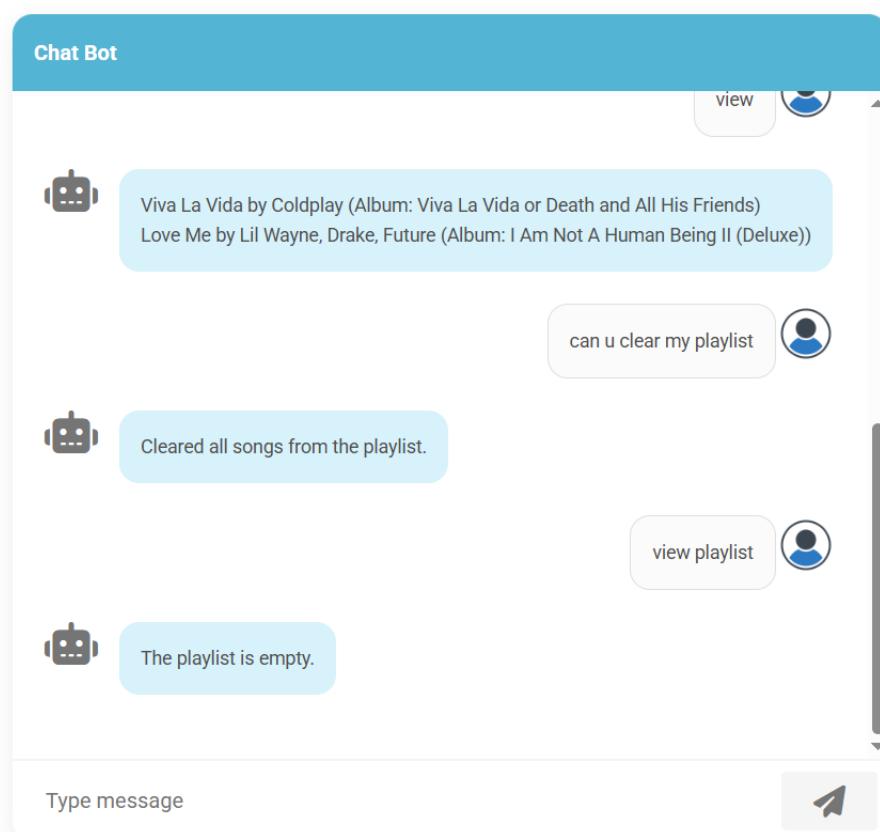


Figura 3.15: Captura de pantalla del chatbot al solicitar vaciar la playlist

### **3. Metodología, Experimentación y Resultados**

---

El resultado final del proyecto es un chatbot musical completamente funcional, que permite al usuario interactuar en tiempo real para realizar tareas como añadir, eliminar, ver o limpiar canciones en una lista de reproducción. Gracias a la integración de modelos de clasificación, el chatbot es capaz de comprender las intenciones del usuario, proporcionando respuestas precisas y apropiadas en función de los mensajes enviados.

## CAPÍTULO 4

# Conclusiones y Trabajo Futuro

---

### 4.1. Conclusiones

#### Consecución de Objetivos

Los objetivos definidos al inicio del trabajo han sido alcanzados satisfactoriamente. Se ha desarrollado un sistema de clasificación basado en Transformers capaz de detectar cuatro intenciones básicas con altas métricas de rendimiento. Además, se ha definido e implementado un flujo de diálogo donde el usuario puede probar el sistema mediante expresiones naturales.

#### Resumen de los Hallazgos

En cuanto al rendimiento global, BERT-base obtuvo ligeramente la mayor precisión ( $\approx 0,99\%$ ), seguido de RoBERTa-base ( $\approx 0,97\%$ ) y T5-small ( $\approx 0,96\%$ ). Además, las curvas de pérdida mostraron que todos los modelos convergieron de manera rápida y estable, sin indicios de sobreajuste, gracias al empleo de `EarlyStoppingCallback` y técnicas de regularización.

#### Implicaciones

Estos resultados demuestran que los modelos encoder (BERT y RoBERTa) son especialmente adecuados para tareas de clasificación de intenciones en interfaces conversacionales, gracias a su robustez y eficiencia en GPU. La flexibilidad generativa de T5-small abre la puerta a sistemas más versátiles que puedan reformular respuestas o adaptarse a nuevos estilos de interacción.

#### Limitaciones

Una de las principales limitaciones es el tamaño y diversidad del corpus. Aunque tenga un balance entre clases, incluye pocas variaciones en la forma de expresar los comandos, por

## **4. Conclusiones y Trabajo Futuro**

---

lo que el modelo puede fallar con frases más complejas.

Por otra parte, dependemos de un entorno con GPU para alcanzar tiempos de entrenamiento razonables. Al usar Google Colab sin suscripción Pro, los recursos (memoria y disponibilidad de GPU) son limitados, lo que nos interrumpía e impedía el entrenamiento de los modelos.

### **Consideraciones Éticas**

Es fundamental considerar las implicaciones éticas de emplear modelos automáticos para interpretar el lenguaje natural en un chatbot musical. Aunque nuestro sistema se centra en comandos de reproducción, cualquier error de clasificación puede alterar la experiencia del usuario o provocar acciones no deseadas (por ejemplo, borrar toda una playlist cuando se quería eliminar solo una canción). Por ello, las decisiones automatizadas deben contextualizarse y acompañarse de revisión humana.

Además, debe garantizarse la privacidad de las interacciones, evitando almacenar información personal innecesaria y permitiendo al usuario revisar y eliminar sus datos si lo desea.

## **4.2. Trabajo Futuro**

### **Extensiones del Estudio**

Actualmente, el asistente se limita a interpretar y clasificar intenciones del usuario, pero podría ampliarse para ofrecer otros servicios relacionados con la música. Una posible extensión podría ser incorporar nuevas funcionalidades como reproducir canciones para así, el sistema pueda realizar acciones directas sobre las canciones.

De este modo, esta nueva funcionalidad abriría las puertas a la incorporación de nuevas intenciones en el modelo. Esto incluiría comandos como reproducir canción", "pausar", "saltar canción" otras funcionalidades relacionadas con la gestión de música. Ampliar las funcionalidades e intenciones permitiría una interacción más completa y mejoraría la experiencia del usuario, llevando el sistema a un nivel más avanzado.

También, ampliar el conjunto de datos para entrenar los modelos sería otra extensión importante. Aunque el dataset actual ha mostrado buenos resultados, incrementar la cantidad de datos y añadir mayor diversidad en las frases de los usuarios podría mejorar la capacidad del modelo para generalizar.

Además, sería interesante incorporar un mecanismo de autenticación y registro de actividad (logging) de usuarios, de modo que cada uno pueda iniciar sesión con sus propias

### **4.3. Planificación Temporal del Trabajo Realizado**

---

creenciales y gestionar una playlist personalizada. De esta forma, cada cuenta mantendría su propia lista de reproducción sincronizada entre sesiones y dispositivos, garantizando la privacidad y continuidad de la experiencia musical.

#### **Investigaciones Adicionales**

Sería valioso investigar cómo los modelos generativos podrían complementar el sistema actual. Aunque en este proyecto se ha utilizado T5 que es capaz de generar un texto de salida, no lo hemos aprovechado y lo hemos adaptado a que generara simplemente la etiqueta correspondiente. Sin embargo, se podría mejorar el sistema si entrenamos este modelo u otro como GPT-3 para generar respuestas más contextuales y coherentes a partir de comandos de texto, mejorando así la experiencia de interacción.

Además, otra idea podría ser probar con modelos multilingües para ampliar el alcance del asistente a diferentes idiomas, haciendo que el sistema sea útil para un público más diverso.

#### **Aplicaciones Prácticas**

Hoy en día, muchas aplicaciones populares como Instagram, WhatsApp, X, y otras plataformas, están empezando a incluir asistentes que utilizan el PLN y el aprendizaje automático para facilitar tareas cotidianas del usuario. Una posible aplicación práctica del sistema es su integración en plataformas de streaming como Spotify o SoundCloud, donde el asistente podría ayudar a los usuarios a gestionar sus listas de reproducción, buscar y reproducir canciones, hacer recomendaciones personalizadas, entre otras funcionalidades.

### **4.3. Planificación Temporal del Trabajo Realizado**

En este apartado se muestra el tiempo empleado en el estudio y resolución de las diferentes partes de nuestro TFG, detallando así las tareas más complejas y las horas globales empleadas en ellas en la [Tabla 4.1](#).

En la tabla se refleja el tiempo dedicado a cada una de las tareas principales. La fase más intensiva fue el desarrollo del sistema de clasificación de intenciones, que incluyó la preparación y tokenización de los datos, el diseño y entrenamiento de modelos, y la experimentación con técnicas de ajuste de hiperparámetros. Esta fase consumió la mayor parte del tiempo del proyecto (150 horas).

El estudio y aprendizaje de las tecnologías necesarias, así como la investigación de modelos preentrenados y su implementación, también fueron tareas clave que requirieron

#### 4. Conclusiones y Trabajo Futuro

---

Tabla 4.1: Planificación temporal del trabajo realizado.

Tarea	Horas
<b>Aprendizaje de los conceptos necesarios para abordar el TFG:</b> - Aprendizaje automático, redes neuronales, Transformers - Transfer Learning, Fine Tuning, PLN, métricas de evaluación	70
<b>Aprendizaje de la tecnología necesaria:</b> - PyTorch, librerías de Transformers, modelos preentrenados - Desarrollo e implementación del chatbot con Flask, Socket.IO y node.js	70
<b>Desarrollo del sistema de clasificación de intenciones (ChatBot):</b> - Preprocesamiento y tokenización de datos - Experimentación y ajustes con Optuna - Diseño, entrenamiento y optimización de modelos - Diseño de la aplicación	150
Elaboración de la memoria del proyecto	50
<b>Total</b>	<b>340</b>

aproximadamente 70 horas. A pesar de no ser abordadas en profundidad durante el curso académico, estas tareas fueron esenciales para poder implementar correctamente las soluciones de clasificación basadas en Transformers.

Finalmente, el tiempo dedicado a la elaboración de la memoria del proyecto fue de 50 horas, lo que permitió completar y documentar adecuadamente el trabajo.

# Referencias

---

- [1] *Aprendizaje Automático*. Wikimedia Foundation, Inc. 2022. URL: [https://es.wikipedia.org/wiki/Aprendizaje\\_autom%C3%A1tico](https://es.wikipedia.org/wiki/Aprendizaje_autom%C3%A1tico).
- [2] Elastic. *¿Qué es la clasificación de texto?* Consultado el 5 de abril de 2025. s.f. URL: <https://www.elastic.co/es/what-is/text-classification/>.
- [3] Carlos Santana. *¿Qué son las redes neuronales?* Vídeo publicado en el canal *DotCSV*. Youtube. 2020. URL: <https://www.youtube.com/watch?v=MRIv2IwFTPg&list=PL-0gd76BhmcB90jPucsnc2-piEE96jJDQ>.
- [4] Yann LeCun, Yoshua Bengio y Geoffrey Hinton. “Deep learning”. En: *Nature* 521.7553 (2015), págs. 436-444.
- [5] Carlos Santana. *¿Qué es una Red Neuronal? Parte 3 : Backpropagation*. Vídeo publicado en el canal *DotCSV*. Youtube. 2020. URL: [https://www.youtube.com/watch?v=eNIqz\\_noix8&list=PL-0gd76BhmcB90jPucsnc2-piEE96jJDQ&index=4](https://www.youtube.com/watch?v=eNIqz_noix8&list=PL-0gd76BhmcB90jPucsnc2-piEE96jJDQ&index=4).
- [6] Jorge Díaz-Ramírez. “Aprendizaje automático y aprendizaje profundo”. En: *Ingeniería. Revista chilena de ingeniería* 29.2 (2021), págs. 180-181.
- [7] Ian Goodfellow, Yoshua Bengio y Aaron Courville. *Deep Learning*. <http://www.deeplearningbook.org>. MIT Press, 2016.
- [8] Aurélien Géron. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*. 2nd. O'Reilly Media, 2019. ISBN: 9781492032649.
- [9] iArtificial. *Función de pérdida en el entrenamiento de una red neuronal*. Feb. de 2023. URL: <https://iartificial.blog/aprendizaje/funcion-de-perdida-en-el-entrenamiento-de-una-red-neuronal/>.
- [10] Isra Sheikh. *Understanding Cross-Entropy Loss and Its Role in Classification Problems*. Oct. de 2023. URL: <https://medium.com/@l228104/understanding-cross-entropy-loss-and-its-role-in-classification-problems-d2550f2caad5>.
- [11] A Cortez Vásquez, Jaime Pariona Quispe, Ana María Huayna et al. “Procesamiento de lenguaje natural”. En: *Revista de investigación de Sistemas e Informática* 6.2 (2009), págs. 45-54.
- [12] Carlos Santana. *INTRO al Natural Language Processing (NLP) 1 - ¡De PALABRAS a VECTORES!* Vídeo publicado en el canal *DotCSV*. Youtube. 2020. URL: <https://www.youtube.com/watch?v=Tg1MjMIVArc>.
- [13] Dan Jurafsky. *Speech & language processing*. Pearson Education India, 2000.
- [14] Tomas Mikolov et al. *Efficient Estimation of Word Representations in Vector Space*. DOI: [10.48550/ARXIV.1301.3781](https://arxiv.org/abs/1301.3781). URL: <https://arxiv.org/abs/1301.3781>.
- [15] Jeffrey Pennington, Richard Socher y Christopher D Manning. “Glove: Global vectors for word representation”. En: *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 2014, págs. 1532-1543.
- [16] Ashish Vaswani et al. “Attention is all you need”. En: *Advances in neural information processing systems* 30 (2017).

## REFERENCIAS

---

- [17] Nicola Tonello. “Lecture notes on neural information retrieval”. En: *arXiv preprint arXiv:2207.13443* (2022).
- [18] Jacob Devlin et al. “Bert: Pre-training of deep bidirectional transformers for language understanding”. En: *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*. 2019, págs. 4171-4186.
- [19] Yinhan Liu et al. “Roberta: A robustly optimized bert pretraining approach”. En: *arXiv preprint arXiv:1907.11692* (2019).
- [20] Colin Raffel et al. “Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer”. En: *Journal of Machine Learning Research* 21.140 (2020), págs. 1-67. URL: <https://arxiv.org/abs/1910.10683>.
- [21] Lisa Torrey y Jude Shavlik. “Transfer learning”. En: *Handbook of research on machine learning applications and trends: algorithms, methods, and techniques*. IGI global, 2010, págs. 242-264.
- [22] Mandy. *Fine-tuning a Neural Network explained*. Vídeo publicado en el canal *deeplizard*. Youtube. 2017. URL: <https://www.youtube.com/watch?v=5T-iXNNiwIs>.
- [23] Zhikang Dong et al. “Musechat: A conversational music recommendation system for videos”. En: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2024, págs. 12775-12785.
- [24] S. Srivatsan et al. “Chatbot Song Recommender System”. En: *International Journal of Research Publication and Reviews* 5.4 (2024), págs. 1477-1483. ISSN: 2582-7421. URL: <https://ijrpr.com/uploads/V5ISSUE4/IJRPR24734.pdf>.
- [25] Srishti Gupta, Riya Chaudhary y Komal Sangwan. *Music&me Chatbot Song Recommender System*. GitHub Repository. 2021. URL: [https://github.com/Srishti20022/Music-me-Chatbot\\_song\\_recommendor\\_system-](https://github.com/Srishti20022/Music-me-Chatbot_song_recommendor_system-).
- [26] Taku Kudo y John Richardson. “SentencePiece: A simple and language independent subword tokenizer and detokenizer for Neural Text Processing”. En: *arXiv preprint arXiv:1808.06226* (2018). URL: <https://arxiv.org/abs/1808.06226>.
- [27] Takuya Akiba et al. “Optuna: A next-generation hyperparameter optimization framework”. En: *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*. 2019, págs. 2623-2631.

## APÉNDICE A

# Anexos

---

### A.1. Repositorio de código utilizado

Con el fin de promover la transparencia y facilitar la reproducibilidad de los resultados obtenidos en este Trabajo Fin de Grado, el código desarrollado ha sido cuidadosamente organizado y se encuentra disponible en un repositorio público de GitHub

El repositorio se encuentra disponible en la siguiente URL

<https://github.com/daniellinfon/Ingenieria-Informatica-UHU/tree/main/TFG>